IT University of Copenhagen

Ph.D. Thesis

Optimizing Liner Shipping Fleet Repositioning Plans

Author:
Kevin Tierney

Supervisor:
Assoc. Prof. Rune Møller Jensen
IT University of Copenhagen

Co-Supervisor:
Prof. David Pisinger
Technical University of Denmark

November 20, 2013

Acknowledgements

I feel truly privileged to have been able to work with and be advised by as many great people as I have over the course of my studies and my PhD. I am grateful to you all, and especially to those named below in no particular order.

First, thank you to my supervisor, Rune Møller Jensen, for his great advice, his insights, and for always being a constructive and positive advisor. Thanks also to my co-supervisor, David Pisinger, for his wisdom and his excitement about research.

My time in Copenhagen would not have been the same without friends and colleagues Dario Pacino and Alberto Delgado, who I thank for their comments on my work, as well as for the times we avoided work. Thanks to Berit Dangaard Brouer, Line Blander Reinhardt and Christian Edinger Munk Plum, for their advice and assistance. I am also grateful to the Masters students who have worked with me on project and thesis topics: Tinus Abell, Björg Áskelsdóttir, Adam Britt, and Christian Kroer.

This PhD would definitely not have been possible without the help of Mikkel Muhldorff Sigurd and Shaun Long at Maersk Line. Thanks to Mikkel for guiding the ENERPLAN project and for opening the doors of Maersk to research. Thanks to Shaun, who devoted considerable time to helping Rune and me understand fleet repositioning and the operations of shipping lines.

I owe much gratitude to my masters advisor Meinolf Sellmann, who introduced me to the world of research during my time at Brown University. Meinolf's energy and wits inspired and motivated me, and played a critical role in me continuing on to a PhD. My thanks to the "Optimization Gang" at Brown: Yuri Malitsky, Serdar Kadıoğlu, Carleton Coffrin, Justin Yip, Olga Ohrimenko, Marie Pellau and Lucile Robin. A special thanks to Yuri, who is a hard working collaborator and a great friend.

I am also grateful to Stefan Voß at the University of Hamburg for hosting me during my "research stay abroad", as well as the entire Institut für Wirtschaftsinformatik for welcoming me so warmly to Hamburg. I have been also fortunate to have had amazing co-authors who have not yet been mentioned, and I therefore thank Carlos Ansótegui, Amanda and Andrew Coles, Elena Kelareva, Philip Kilby, and Robert Stahlbock.

I have no doubt I would not be where I am today if it were not for Hans-Peter Bischof's encouragements during my bachelor's degree for me to do a PhD. I thank him for that, and for the study abroad he organized to Osnabrück, Germany that remains a significant milestone of my life. I also thank the members of my PhD committee for their helpful comments that have improved this work.

Finally, and most importantly, I wish to thank my parents for always putting my education first and supporting me in everything I have done, even when moving halfway around the world. I also thank my wife, Christine, for her love and support, especially throughout my PhD, which is a daunting enough task without having a long distance relationship as well. We've endured weeks and months of separation and the stress of my deadlines, and I thank you for being with me through it all.

Abstract

With the incredible growth of containerization over the past half century, shipping lines and ports are facing increasing challenges in ensuring that containers arrive at their destinations on time and on budget. This dissertation addresses several critical problems to the operations of shipping lines and ports, and provides algorithms and mathematical models for use by shipping lines and port authorities for decision support. One of these problems is the repositioning of container ships in a liner shipping network in order to adjust the network to seasonal shifts in demand or changes in the world economy.

We provide the first problem description and mathematical model of repositioning and define the liner shipping fleet repositioning problem (LSFRP). The LSFRP is characterized by chains of interacting activities with a multi-commodity flow over paths defined by the activities chosen. We first model the problem without cargo flows with a variety of well-known optimization techniques, as well as using a novel method called linear temporal optimization planning that combines linear programming with partial-order planning in a branch-and-bound framework. We then model the LSFRP with cargo flows, using several different mathematical models as well as two heuristic approaches. We evaluate our techniques on a real-world dataset that includes a scenario from our industrial collaborator. We show that our approaches scale to the size of problems faced by industry, and are also able to improve the profit on the reference scenario by over US$14 million.

This dissertation also addresses the topic of inter-terminal transportation (ITT), which involves minimizing the delay experienced by containers being transported between terminals in a port under varying infrastructure configurations and material handling equipment properties. Minimizing the delay of ITT is an important problem in the strategic planning of new ports and port expansions, and one that has not yet been addressed in an optimization based approach. We provide the first mathematical model of ITT and show how the model can be used to provide critical information to port authorities on two real ports, the port of Hamburg, Germany, and the Maasvlakte area of the port of Rotterdam, Netherlands.

Finally, this dissertation gives a polynomial time algorithm for an open problem from the container stowage literature, the capacitated k-shift problem with a fixed number of stacks and stack heights, providing an answer to a 13 year old theoretical question in the container stowage domain.

Contents

Chapter 1

Introduction

Situated at the heart of global trade, liner shipping networks transported roughly 1.5 billion tons of cargo on over 5,000 container vessels in 2012 [154]. Liner shipping networks primarily transport containers, which are steel boxes of one of several standardized dimensions that can be filled with goods for shipment. Despite a short period of decline in 2009, container shipment volumes have been steadily increasing ever since the container's invention in the 1950s [94, 154], and this trend looks set to continue.

As the use of containers has increased, so too has the challenge of getting containers to their destinations on time and on budget. Modern shipping lines have networks that span the entire globe and use hundreds of container ships to move millions of containers. In order to deal with the complexity and scale of the problems they are facing, shipping lines are turning to algorithmic approaches and decision support systems that assist their employees in making difficult decisions.

The research community has identified and modeled several well-known optimization problems that attempt to alleviate the challenges that the liner shipping industry faces (see, e.g., [23, 24]). At the strategic level, shipping lines must create the routes in their network to satisfy their customers' demands, a problem that despite a number of years of research is still too difficult to solve for most realistic sized problems [4, 122]. At the tactical level, container ships are assigned to services in an existing shipping network in order to minimize the sailing costs while keeping customer's demands in consideration [120]. And, finally, at the operational level, shipping lines must determine vessels' sailing speeds [44, 163], which vessels carry which containers, and manage disruptions [18].

Through our collaboration with Maersk Line, currently the world's largest container shipping line, we found that a key operational problem has not yet been addressed in the liner shipping literature. Liner shipping networks are regularly adjusted in order to handle fluctuations in seasonal demand, as well as to adjust the network to changing macroeconomic conditions. These adjustments can involve adding new routes to the network, removing unprofitable routes, or expanding/contracting existing routes. Container ships must be *repositioned* within the network to facilitate these changes. Repositioning involves sailing vessels between routes in a liner shipping network in or-

der to minimize sailing costs, port fees, and disruptions to container flows. During repositioning, ships may undertake a number of cost saving activities, such as bringing empty containers to places where they are needed or temporarily taking over operations on existing services.

Repositioning a ship can cost upwards of a million US dollars, and hundreds of ships are repositioned each year by the world's shipping lines. Finding plans of activities that minimize the cost of repositioning, while avoiding the disruption of cargo flows in the network, has the potential to save shipping lines significant amounts of money. Currently, ship repositioning plans are created by hand by employees of shipping lines. Furthermore, since one of the main cost components of repositioning ships is bunker fuel, optimizing vessel activities can also help shipping lines reduce their CO_2 output and become more sustainable. This dissertation considers the following core question:

> *Can an algorithm be developed to create real-world realizable liner shipping fleet repositioning plans within a reasonable amount of CPU time?*

Fleet repositioning involves a number of side constraints. We aim to model key constraints that determine a plan's real-world feasibility, such as constraints on the sailing speed and capacity of ships, and by ensuring the routes determined for ships adhere to a number of liner shipping specific constraints. Our goal is to generate repositioning plans in under an hour in order to allow repositioning coordinators to use our algorithms in an interactive fashion in a decision support system. Since even a small improvement in the objective function can translate to tens of thousands of dollars in cost savings or increased revenue, our algorithms should find repositioning plans that are within a few percent of the optimal solution.

We show that fleet repositioning problems can be solved in many cases to optimality with or without cargo flows on a dataset based on real-world data from our industrial collaborator. In addition, we compare our solution approaches for fleet repositioning to a real world scenario from Maersk Line in 2011. Our solution methods are able to find a repositioning plan with a profit of \$32.1 million[1], which is roughly \$14 million higher than the profit of the actual repositioning that was carried out by Maersk Line.

1.1 Approach and Contributions

We model fleet repositioning with and without cargo flows, and call the general problem of repositioning the liner shipping fleet repositioning problem (LSFRP). We test both models on datasets based on real scenarios from our industrial collaborator and crafted scenarios using industrial data.

We first model the problem without cargo flows in order to focus on minimizing the cost of the activities chosen during repositioning, and call this problem the no-cargo LSFRP (NCLSFRP). This problem has a number of simplifications of the overall problem with cargo flows that we present in detail in Chapter 5. A number of activities,

[1]All monetary units in this dissertation are US dollars.

such as sailing a vessel, have time-dependent task costs, i.e., the cost of performing an activity is dependent on the action duration. The NCLSFRP is difficult to solve due to these costs as well as because of the interactions of vessels in choosing which activities each vessel should perform. Since the problem contains both scheduling and routing components, it is not clear a priori which types of solution method will solve it most effectively.

We present four different solution approaches and describe the trade-offs between them, starting with an automated planning PDDL model. Since existing automated planning techniques have difficulties with time-dependent task costs, we introduce a novel planning technique called linear temporal optimization planning (LTOP), which combines the well-known partial-order planning paradigm with linear programming. We model the NCLSFRP in the LTOP framework and show that planning and optimization need not be mutually exclusive endeavors. We also provide a mixed-integer programming (MIP) model and constraint programming (CP) model of the NCLSFRP, showing that while the MIP has a difficult time with the scheduling aspects of the problem, CP is able to outperform all other approaches, albeit with the least extensible model. Our PDDL, LTOP and MIP models were first published in [147], and our CP model in [83].

We then move to a model of the LSFRP with cargo and equipment flows. In addition to the challenge of optimizing activities with time-dependent task costs, we incorporate a multi-commodity flow problem to handle cargo and equipment. We also include a more realistic view of certain problem activities than in the NCLSFRP. We create a graph based representation of the problem that includes a number of key problem constraints and activities, such as sail-on-service opportunities, within the graph itself. We use this graph in all of our solution methods for the LSFRP with cargo and equipment flows.

We first model the problem as a MIP using an arc flow formulation and show that, while small instances can be solved to optimality, larger instances prove challenging for the MIP. We also introduce a node flow model for a special case of the LSFRP in which all activities have a fixed start and end time. On those instances that conform to this simplification of the LSFRP, we are able to find optimal solutions to nearly every instance in our dataset, including several instances that the arc flow model cannot solve.

Furthermore, we introduce a path based model which is solved using a column generation procedure. This approach solves even more problems that neither the arc flow or node flow models can solve. This model iterates between a master problem which selects vessel paths through the graph, and a sub problem that generates paths. Since column generation can only be used for solving linear programs (rather than mixed-integer programs), the path based model is only guaranteed, from a theoretical perspective, to find a lower bound to the overall LSFRP. However, our path based model finds solutions with integer values for 95% of our dataset, meaning it finds the optimal solution.

Finally, we also introduce two heuristic approaches based off of simulated annealing (SA) [85] and late acceptance hill climbing (LAHC) [19], respectively. Although our

heuristic approaches are not always able to find optimal solutions, they are effective at providing good solutions quickly. We presented our work on the arc flow model of the LSFRP in [149], the node flow model in [150], and on solving the LSFRP with heuristics in [144] and [143].

The contributions of this dissertation are as follows:

1. A planning model of the NCLSFRP.
2. A formalism of temporal optimization planning and its linear instantiation, linear temporal optimization planning.
3. A linear temporal optimization planning model of the NCLSFRP.
4. A mixed-integer programming model of the NCLSFRP.
5. A constraint programming model of the NCLSFRP.
6. A graph construction and arc flow mixed-integer programming model of the LSFRP with cargo flows.
7. A simulated annealing and late acceptance hill climbing approach to the LSFRP with cargo flows.
8. A node flow mixed-integer programming model of a specialized version of the LSFRP with cargo flows.

We therefore answer the core question of this thesis with "yes"; that is, an algorithm can be developed to create liner shipping fleet repositioning plans for a decision support system within a reasonable amount of CPU time.

1.2 Secondary Thesis Objectives

In addition to fleet repositioning, this dissertation considers two secondary thesis questions from the liner shipping and container port optimization domains.

1.2.1 Inter-Terminal Transportation

Container ports are facing new challenges as more and larger vessels increase container traffic around the world. The world's container port throughput in 2011 reached a new record of 572.8 million TEU[2] [154]. In order to handle such high volumes of containers, ports, like shipping lines, are turning to automated decision support to assist employees in their daily tasks. The research community has focused on solving a number of key problems that ports face in order to increase their efficiency in the face of increasing container flows. In particular, there has been an emphasis placed on modeling the internal processes of container terminals, including automated guided vehicle (AGV) and automated lift vehicle (ALV) routing (see [6]), berth scheduling (see [13]), crane split optimization (e.g., [116]), and yard storage optimization (e.g., [21, 93]).

[2]TEU stands for "twenty-foot equivalent unit", and is explained in detail in Section 2.1.

However, most ports consist of multiple terminals, and many containers travel from one terminal to another as they are transported to their final destination. *Inter-terminal transportation* (ITT) is concerned with the movement of containers between terminals as they are transshipped or transferred between different transportation modes. New ports include more and more terminals to keep up with demand, and old ports are expanding the number of terminals present, such as the new Maasvlakte II area of the port of Rotterdam, Netherlands [119]. Ports require decision support to help them investigate ITT using various configurations of vehicles and vehicle types, as well as under varying types of infrastructure connections between the terminals. Several simulation studies [41, 110, 111] have been proposed to investigate ITT, but no mathematical model exists in the literature for finding minimal cost flows of containers between terminals. With such a model, ports could analyze different infrastructure and vehicle options, allowing them to make their port more competitive and reduce delays in containerized transport. To this end, this dissertation examines the following question:

> *Can a general model of inter-terminal transportation be developed to minimize delay that handles arbitrary types of material handling equipment and transportation infrastructure, and be solved to optimality within an hour of CPU time?*

We provide a novel mathematical model using a time-space graph to minimize the delay in delivering the containers in a set of inter-terminal demands. Our model handles a variety of infrastructure configurations, as well as varying material handling equipment, such as automated guided vehicles (AGVs), automated lift vehicles (ALVs), multiple-trailer systems (MTSs) and barges. We show what kind of information the model can provide to port and terminal operators on a dataset of instances representing the ports of Rotterdam, Netherlands and Hamburg, Germany.

1.2.2 Container Stowage

We also solve an open problem from the field of container ship stowage, which concerns itself with the safe and efficient loading of containers onto container ships. The stowing of containers on container ships is a difficult problem that has to take into account a number of constraints to ensure the stability of the vessel and the safe stowage of containers (see, e.g., [113]). One of the main objectives of container stowage is the minimization of *shifting*, in which containers must be removed from the vessel and are placed back into the vessel in order to unload a container. Although there are a number of successful approaches for solving container stowage problems, such as the decomposition approach of Wilson and Roach (2000) [165] and the recent approaches of Delgado, et al. (2012) [38] and Pacino, et al. (2011) [114], little is known about the theoretical complexity of stowage planning. Avriel, et al. (2000) [10] introduced the *zero shift problem*, in which they model a simplified version of stowage planning that minimizes the number of shifts that must be performed in a set of container stacks over some number of time points. However, they were unable to find a proof of

the complexity of the capacitated k-shift problem, which occurs when there is a fixed number of stacks with fixed heights. We provide an algorithm and proof of runtime and correctness showing that the capacitated k-shift problem with fixed stacks and fixed stack heights is solvable in polynomial time.

1.3 Outline

The outline of this dissertation is as follows.

Chapter 2 We provide background on containerized shipping both at sea and on land. We describe the components relevant to fleet repositioning, inter-terminal transportation and the capacitated k-shift problem, including liner shipping networks, container ships, and port operations.

Chapter 3 In this chapter, we describe liner shipping fleet repositioning in detail. We introduce all of the cost saving (and revenue earning) activities present in fleet repositioning, as well as describe a case study performed with our industrial collaborator.

Chapter 4 We provide background information for the several different solution approaches used in this dissertation, including automated planning, mixed-integer programming and constraint programming.

Chapter 5 We model and solve a simplification of fleet repositioning that ignores cargo flows, but focuses on other difficult aspects of the problem, such as time-dependent task costs. We use automated planning, an automated planning and linear programming hybrid called linear temporal optimization planning, mixed-integer programming and constraint programming to find optimal solutions on a dataset of instances modeled on a real repositioning scenario.

Chapter 6 This chapter examines the LSFRP including cargo and equipment flows using a specialized graph. We solve the problem by formulating models based on an arc flow, path generation, and a node flow. In addition, we present two heuristic solution approaches, simulated annealing and late acceptance hill climbing, in order to solve problems that are too large for the optimal approaches.

Chapter 7 We shift our focus in this chapter to port operations and describe inter-terminal transportation. We present a novel mathematical model that minimizes the delay of container transports between port terminals.

Chapter 8 We present a polynomial time algorithm for the capacitated k-shift problem with a fixed number of stacks and fixed stack heights in this chapter. Our proof of correctness and runtime provide an answer to a problem that has been open since the year 2000.

Chapter 2

Containerized Shipping

Before Malcolm McLean invented the shipping container in 1956, transporting cargo by sea was a long and arduous process. Longshoremen unloaded trucks and trains and packed ships full of boxes and bags of goods, and then unloaded them again at their destination, a process that could take several days for just one ship. Shipping goods was not only expensive, but cargo often had a tendency to go missing or be damaged along the way to its destination [35, 94]. Containers, which are large steel boxes in which goods are placed, revolutionized seaborne trade, allowing cargo to be securely, safely and quickly handled and transported to destinations around the world. With containers, cargo can be quickly transferred not only to and from ships, but also between various modes of transportation, such as between trains, trucks and ships. Containers have significantly lowered the costs of maritime trade, and costs continue to decline with the increasing size and efficiency of seagoing vessels [154].

Liner shipping encompasses most of the world's seaborne containerized shipping, in which specially built seagoing vessels carry thousands or even tens of thousands of containers on a regular schedule between ports. The "liner" in liner shipping refers to the way vessels follow one another in a line-like fashion along a route. The fixed schedule of liner shipping is a key selling point for the liner shipping industry, and is what differentiates it from *tramp* or *industrial* shipping. A common analogy to understand the difference between liner shipping and tramp shipping is that the regular schedule of liner shipping is similar to that of a bus or rail network, except the periodicity is generally weekly, rather than every few minutes or hours. In contrast, in tramp shipping vessels act more like taxis, carrying cargo without a fixed schedule or route [25, 125]. In both tramp and liner shipping, the goal of shipping firms is to maximize their profit, whereas in industrial shipping the goal is to minimize costs.

In the remainder of this chapter, we will discuss the innovation that makes containerized shipping possible, the container, followed by an overview of liner shipping networks, which connect the ports of the world. Finally, we look at the components of a container port, the interface between the land and the sea.

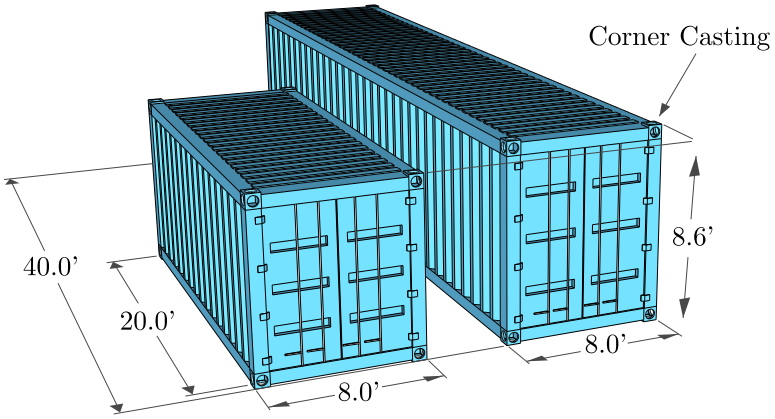


Figure 2.1: A twenty foot and a forty foot container, adapted from [113].

2.1 Containers

The centerpiece of containerized shipping is the intermodal container, a rectangular steel box with castings on all of its corners to allow the safe stacking of multiple containers. Standard containers have a width of 8 feet, a height of 8 feet 6 inches and are either 20 or 40 feet long, as depicted in Figure 2.1. A variety of variations on these standard containers exist, such as high cubes and 45 foot length containers, which are outside the scope of this work.

Quantities of containers are measured in either the *twenty-foot equivalent unit* (TEU) or the *forty-foot equivalent unit* (FEU or FFE). A TEU represents a single twenty foot container, and one FFE is equal to two TEU (i.e. two twenty-foot containers or one forty-foot container). Even though there are a number of different sizes of containers, not all of which have a length divisible by 20, their quantities are measured in TEU or FFE nonetheless.

Numerous types of containers exist to fit the varied needs of shippers, such as tank containers for liquids, open top containers for tall cargo, and refrigerated containers for cargo that needs to be chilled, to name a few. Many non-standard containers require special handling and storage procedures to ensure that their contents are not damaged and to ensure the safety of the vessel and its crew.

This dissertation focuses primarily on *dry* and *reefer* (i.e. refrigerated) containers. Dry containers are standard 20 or 40 foot containers requiring no special handling and can be stored anywhere on a vessel. Reefer containers, which can also be either 20 or 40 feet in length, have an integrated refrigeration unit and therefore require access to an electrical outlet on board a vessel. Most vessels only have a limited number of locations where an outlet is available, meaning the number of reefer containers that can be transported by a vessel tends to be less than the number of dry containers.

2.2 Liner Shipping Networks

The goal of a *liner shipping network* is to facilitate the transport of containers between ports. A liner shipping network is defined by a published, periodic schedule, that determines when vessels visit ports. The periodicity of the schedule is central to liner shipping, in that seagoing vessels visit ports on a weekly or bi-weekly basis, at the same time each week. Liner shipping networks can span the entire globe or be constrained to a specific geographic region, depending on the shipping line and its customers. The overall structure of a liner shipping network takes the form of either a hub-and-spoke, a direct routing approach or a mix of hub-and-spoke and direct routing [70, 109].

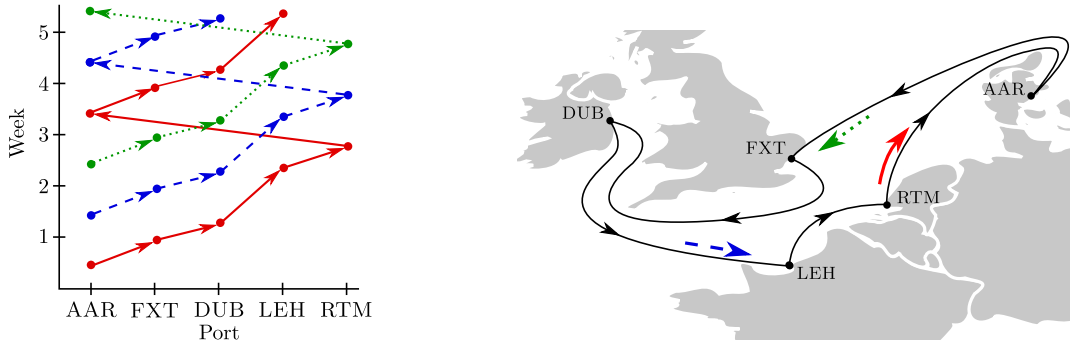
2.2.1 Services

In order to achieve a weekly or bi-weekly regularity, vessels are sent on cyclical routes called *services*[1]. Services are cyclical routes that visit ports on a regular, usually weekly, schedule. A weekly service has as many vessels as the number of weeks a vessel requires to complete a single rotation of the service. That is, a service with a rotation duration of 5 weeks requires 5 vessels in order to maintain a weekly frequency. Each vessel is assigned a *slot* on the service, thus, there are as many slots on the service as there are vessels. A slot can be viewed as containing the specific schedule for each vessel. In other words, while the schedule for a service specifies that a particular port is visited, for example, on Mondays at 17:00, the schedule for a slot provides the exact day that the vessel in that slot should visit the port (i.e., April 22, 2013 at 17:00). For a service with a weekly regularity, the vessel in the subsequent slot would visit the port on April 29, and the vessel in the previous on April 15. Each port visited on a service slot is referred to as a *call*. Shipping lines assign each call an arrival and departure time. Between the arrival and departure time, the vessel sailing on the service loads and unloads containers. Each slot contains multiple *legs*, and each leg begins at a port call and ends at the next scheduled call.

An example service

Figure 2.2 shows an example liner shipping service in the North Sea, consisting of a time-space graph in Figure 2.2a and a geographical representation of the service in Figure 2.2b. The three week long service contains three vessels to visit its 5 port calls. Notice the cyclical structure of the service, in which a vessel, upon reaching AAR from RTM, continues sailing to FXT. Additionally, all three vessels are sailing at the same time with a temporal spacing of a week. The positions of each vessel, shown as red, green and blue arrows, at the start of week three of the service is shown in Figure 2.2b. Table 2.1 gives the schedule of the example service in Figure 2.2. The schedule displays

[1] In shipping parlance, services are sometimes called "strings". We avoid this term to prevent confusion with programming terminology.



(a) Time-space graph of a service.

(b) Geographical representation of a service.

Figure 2.2: An example liner shipping service connecting several nations of Northern Europe with the duration of a single rotation lasting three weeks. The position of each of the three vessels required for the service in (b), represented by red, blue, green arrows, corresponds to week three in (a).

Port	Arrives		Departs		Days
AAR	Thursday	01:00	Thursday	08:00	0
FXT	Sunday	12:00	Monday	02:00	3
DUB	Tuesday	23:00	Thursday	03:00	5
LEH	Wednesday	08:00	Wednesday	18:00	13
RTM	Saturday	06:00	Saturday	14:00	16

Table 2.1: A time table showing the arrival and departure time of each call on the example service from Figure 2.2.

the day of the week and the time of day that each vessel arrives and departs each port, along with the number of days since the start of the rotation at AAR.

Service structure

Due to trade imbalances on some services, the amount of containers carried on each leg of a service may vary greatly. Many services can be split into two components, the *headhaul* and the *backhaul*. On the headhaul, containers are carried from areas of production to areas of demand, and vessels are generally at or near capacity. For example, vessels traveling from Asia to the US or Asia to Europe are on the headhaul of their service. On the backhaul, vessels are returning to areas of production and are relatively empty. On some services, a single port separates the headhaul from the backhaul. Such ports are called *turn ports*, and hold a special significance for optimizing the movement of container vessels since the vessel tends to offload nearly all of its containers and is close to empty before it loads more containers. Turn ports offer opportunities to move vessels on to and off of services that avoid disrupting the flow of cargo.

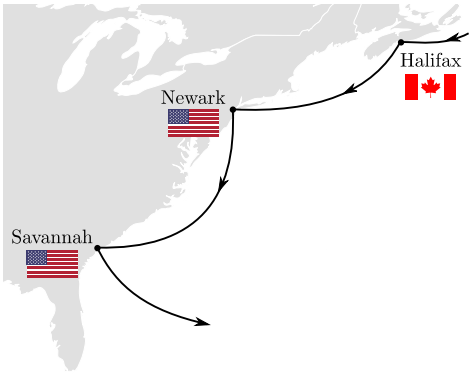


Figure 2.3: An example service along the US and Canadian east coast.

Pick-up	→ Delivery	US Flag	Non-US Flag
Halifax	→ Newark	✓	✓
Halifax	→ Savannah	✓	✓
Halifax	→ Non-US Port	✓	✓
Newark	→ Savannah	✓	✗
Newark	→ Non-US Port	✓	✓
Savannah	→ Non-US Port	✓	✓

Table 2.2: Legal and illegal cargo pick-up and delivery pairings due to US cabotage laws.

2.2.2 Network structure

Modern liner shipping networks are generally composed of a hub-and-spoke-like network, with several ports serving as hubs between multi-port *trunk* services and regional *feeder* services. Liner shipping networks do not conform exactly with a hub-and-spoke design, and are often a mix of direct connections and feeder services.

Although there is no strict definition of trunk services, they generally connect ports between geographic regions and have a long duration, requiring four or more weeks to complete a rotation. Trunk services can connect hubs to each other, connect hubs to distant markets, such as a hub in Asia to the ports of Northern Europe, or connect large ports to each other, such as Shanghai, China and Oakland, USA. Panamax vessels or larger usually sail on trunk services. Feeder services, in contrast, are the spokes of the network, and ferry cargo from large hub ports to smaller ports that do not merit visits by the trunk services. Small vessels, nearly always Panamax or smaller, travel on feeder services. Small vessels are more economically viable on such routes [109] due to their lower bunker consumption, which we will explore in Section 2.2.3.

Transshipments

Containers are moved between services through *transshipments*, in which a container is offloaded from a vessel, temporarily stored at a container terminal, and then loaded onto another vessel. Transshipments allow liner shippers to provide service to smaller ports, as well as aggregate cargo along the trunk services in their networks. Containers may undergo several transshipments on their way to their destinations. A number of key ports act as the central transportation hubs for many shipping lines, such as Rotterdam, Netherlands; Singapore; Tanjung Pelepas, Malaysia; and Hong Kong, China. Rotterdam is a local hub for cargo traveling to Europe, and Singapore, Tanjung Pelepas, and Hong Kong collect cargo from ports around Asia before it is forwarded on vessels sailing to the rest of the world.

Figure 2.4: A quay crane services the post-Panamax vessel Maersk Edinburgh at the Burchardkai Terminal in the port of Hamburg, Germany.

Sailing restrictions

Liner shipping networks must be carefully constructed in order to avoid violating *cabotage restrictions*, which are laws that prohibit the shipment of domestic cargo with a foreign flagged vessel. Note that a vessel's *flag* denotes the country where it is registered. A vessel registered in one country is therefore not allowed to pick up cargo in a different country with cabotage laws and deliver that cargo to the same country. Although this is a simplification, as cabotage laws vary from country to country and have many details, this understanding of cabotage laws is sufficient for the purposes of this dissertation. Figure 2.3 shows an example service from Halifax, Canada to Newark and Savannah USA, and Table 2.2 lists several possible cargo routes along the service. The table lists which pick-ups and deliveries would be allowed for a US flagged vessel versus a non-US flagged vessel, due to cabotage laws. Both vessels are allowed to carry cargo in most cases. However, when sending cargo from Newark to Savannah, a US flagged vessel must transport the cargo. Also note that in the case where cargo is being shipped from Halifax to Savannah, if the cargo were to be transshipped in Newark to a non-US flagged vessel it would be violating cabotage laws.

Shipping lines are further restricted in where their vessels may travel due to laws preventing certain types of cargo, such as governmental aid or military equipment, from traversing the ports of certain nations. For example, a ship carrying US military equipment should not travel through, say, Iran or North Korea. Even though these restrictions are taken into account during the design of a network, they must be considered when dealing with disruptions or changes in the network.

2.2.3 Vessels

Container vessels are classified into several categories based on the length and width of the vessel [124]. The Panamax class of vessels describes any vessel able to fit through the locks of the Panama canal. Panamax vessels have a maximum length of 294.13

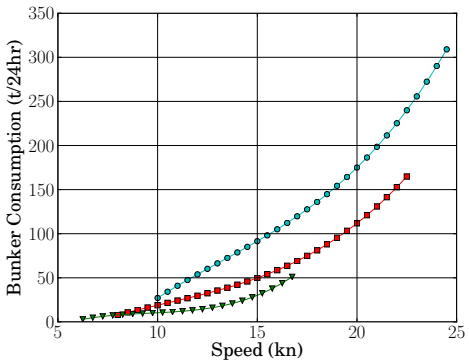


Figure 2.5: Bunker consumption in metric tonnes per 24 hours for a feeder vessel (green, triangles), Panamax vessel (red, squares), and a post-Panamax vessel (cyan, circles). Data source: Anonymized data from Maersk Line.

meters, beam (width) of 32.31 meters and draft (depth in the water) of 12.04 meters. Panamax vessels, depending on their build specifications, can carry up to 5,000 TEU [124]. Post-Panamax vessels are vessels that exceed the maximum vessel dimensions of the Panama canal in one or more dimensions, and can therefore not travel through the canal. Figure 2.4 shows the post-Panamax vessel Maersk Edinburgh at call in Hamburg, Germany. The vessel is 366 meters long and can carry up to 13,092 TEU [123]. Feeder vessels are small vessels used on regional routes, such as within Northern Europe or the Mediterranean Sea, that carry around 1,000 to 2,000 TEU.

Vessels consume a thick, heavy fuel called *bunker fuel*, usually heavy fuel oil (HFO) [108], for propulsion. The amount of bunker consumption varies with the speed of the vessel. Vessels have a minimum speed, and traveling at a speed lower than this value could cause damage to the engine. Recently, shipping lines have begun *slow-steaming*, in which the speed of vessels is reduced in order to reduce bunker fuel consumption [73, 101]. Figure 2.5 shows approximate bunker consumption curves for three vessel classes from Maersk Line's fleet[2] in metric tons of fuel per 24 hours given the speed of the vessel in knots. The green line with triangles shows the curve for a roughly 1000 TEU vessel used for feeder services, the red line with squares gives the consumption curve for a Panamax vessel that can carry around 4500 TEU, and the cyan line with circles displays the bunker usage of a post-Panamax vessel that can carry over 10,000 TEU. Slow-steaming is especially valuable for Panamax and larger vessels. For example, traveling at its minimum speed from Shanghai, China to Los Angeles, USA, the post-Panamax vessel from Figure 2.5 requires only 21.5% of the fuel it would require traveling at full speed, albeit the journey takes over twice as much time. For the Panamax vessel, the savings are even larger. It requires only 13.5% of the fuel it would need to travel at its maximum speed, but its journey, as in the case of the post-Panamax vessel, becomes much longer.

Slow steaming and the bunker consumption of vessels has been the topic of several

[2]The curves are only approximate in order to protect the confidentiality of Maersk Line's data.

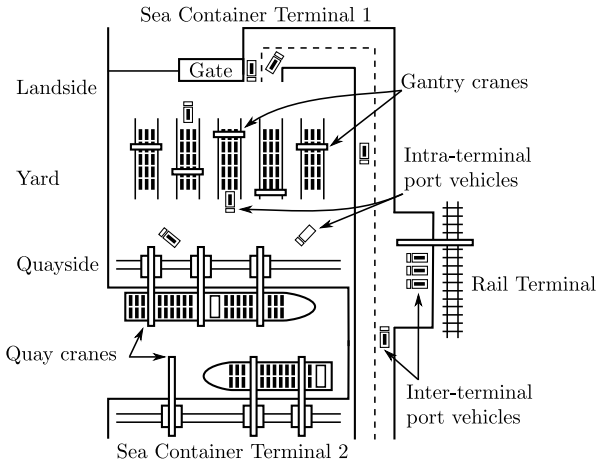


Figure 2.6: A port with two sea terminals and a rail terminal.

recent works that have emphasized the importance of including a realistic view of bunker consumption. Optimizing the sailing speed of a vessel on an existing service can result in bunker fuel consumption reductions of up to 25% [44], and optimizing several services also results in fuel reduction [163].

Bunker consumption can be approximated with a third-degree polynomial [163], but can also be modeled with a linear approximation [17] in order to harness linear programming and mixed-integer linear programming solvers for solving problems with variable vessel speed.

2.3 Ports and Container Terminals

A container's journey through a liner shipping network starts and ends at a port. Ports consist of one or more container *terminals*, at which containers are loaded and unloaded from ships and stored in the yard or transferred to land based transportation. Terminals can be separated into a *quayside*, *yard*, and a *landside* [139]. The quayside is the interface between the terminal and the liner shipping network, whereas the landside is the juncture between the terminal and hinterland transportation or transportation between terminals. Between the quay and land is the yard, in which containers are stored until the next leg of their journey is ready to begin.

Figure 2.6 shows an example port containing two sea terminals and a rail terminal. For incoming cargo, quay cranes unload containers from ships and place them on intra-terminal container handling vehicles, such as automated-guided vehicles (AGVs) or multi-trailer systems (MTSs), or the containers are positioned along the quay and are retrieved by automated-lift vehicles (ALVs), which have the ability to load and unload containers without the assistance of an external crane. The vehicles then bring the containers to the container stacks in the yard. In this example, rail mounted gantry cranes retrieve the containers from the vehicles and place them in the stacks, although

in some ports straddle carriers take this role [139]. At the landside of the terminal, vehicles, such as trucks, retrieve containers to bring to customers. Some containers may also be brought to other terminals in the port, such as other sea terminals or rail terminals. This is called *inter-terminal transportation* and is the subject of Section 7.

Container terminals earn money through moving containers on and off of vessels and storing them in the yard. Container terminals earn money for each *move* they perform, which is the transfer of a single container, regardless of size, on or off a vessel. Rates per move are usually in the range of \$50 – \$300 depending on the port and whether the container is empty or full (empty container moves tend to be cheaper than full moves). Transshipments tend to cost more than a single move, but are cheaper than performing both a load and an unload move. Ports also earn money through yard storage fees, although transshipments usually include a short period of yard storage in the price.

Chapter 3

Liner Shipping Fleet Repositioning

As production and demand wax and wane across the world, liner shipping networks, must be constantly adjusted due to seasonal shifts in demand, and to ensure that they are compatible with the current state of the world economy. Shipping lines must add, remove and modify existing routes in their network in order to keep it relevant for their customers. When a shipping line decides to change its network, vessels must be moved between services in order to realize the changes. The process of moving vessels in such a manner is called *repositioning*. Shipping lines perform one or two large network changes requiring repositionings each year, along with several smaller repositionings throughout the year.

Shipping lines must reposition vessels in order to stay in business, but repositioning vessels is generally not a revenue earning endeavor. Shipping lines therefore seek to minimize the cost of repositionings by reducing fuel consumption as well as avoiding, as much as possible, the disruption of normal operations of the network. Repositioning a single vessel can cost up to, and sometimes over, a million US dollars, according to our industrial collaborator Maersk Line. Hundreds of repositionings are undertaken every year by the world's shipping lines, yet the problem of optimizing the movements of repositioning vessels has been scarcely studied.

At first glance, vessel repositioning sounds like a standard vehicle routing problem (VRP) in which vessels must be moved from one location to another. On the contrary, the structure of liner shipping networks as well as liner shipping specific opportunities result in a problem quite different from the well known vehicle routing variants. In fact, a plethora of activities exist for repositioning vessels to save money or avoid container flow disruptions, such as utilizing existing routes of the network, carrying empty containers, varying the sailing speed, or re-scheduling certain parts of a route, to name a few. Additionally, in repositioning problems there is no requirement of visiting every port call. Rather, doing so would be extremely expensive and is discouraged.

Time-dependent task costs [83] complicate fleet repositioning problems. Time-dependent task costs involve activities in which an activity's cost varies with its duration. Such activities are found in problems like the vehicle routing problem with soft time windows (VRPSTW) (e.g., [121]) and ship scheduling with time-varying

draft [81, 82]. Time-dependent task costs create difficult to solve trade offs for optimizers, in that changing the duration of an activity may improve the cost of the activity itself, but such changes also mean that the opportunities available after the activity completes are different. Thus, improving changes in the objective for a single activity could have a detrimental effect on the objective as a whole, in that later activities are now more expensive. Furthermore, predicting the consequences of adjusting the duration of a single activity can be difficult to predict, due to the propagation of this change to later activities.

We assume that the vessels involved in a repositioning are selected by a repositioning coordinator, rather than chosen by an optimization model. This helps to prevent side effects in the liner shipping network, and any consequences of the choice of vessels involved can be handled directly by the repositioning coordinator.

In this chapter we describe all of the details of repositioning problems, starting with an overview and definition of repositioning in Section 3.1. The procedures by which vessels leave and enter services, and thus start and end a repositioning, are described in Section 3.2, followed by a discussion of the activities that vessels may undertake during a repositioning in Section 3.3. We go on to describe a real world repositioning in the Pacific conducted by our industrial collaborator, Maersk Line, in Section 3.4. We conclude the chapter in Section 3.5 with a discussion of well known problems from the literature that are similar to repositioning problems.

3.1 Repositioning Overview

The goal of repositioning vessels is to carry out one of two types of network changes: *i*) to create a new service, or *ii*) to expand an existing service. For both of these network changes, vessels need to be brought from somewhere else in the network, which may require shutting down a service or reducing the number of vessels on a service. We can actually view the expansion of an existing service as the creation of a new service in which several of the vessels happen to already be on schedule. This allows us to have a unified understanding of repositioning in which multiple vessels must leave their initial services in order to travel to a new, goal service that is being created. Vessels must occasionally undergo repairs or routine maintenance, which also requires repositioning. We view this as a special case of the above repositioning goals, with repairs being an activity that must be performed during the repositioning (with some side constraints). Nonetheless, we do not currently model repairs in this work, and thus do not describe repair procedures in detail.

Figure 3.1 shows a geographic view of a repositioning, in which a vessel from the red service (right) and a vessel from the green service (middle) are repositioned to start the new blue service (left). A potential path of the vessel from the red service is shown as a solid black line, and a dashed line shows a potential path for the vessel from the green service. Note that there are a number of different options, and the repositioning we show is only one possibility of many. The vessels sail from their initial services

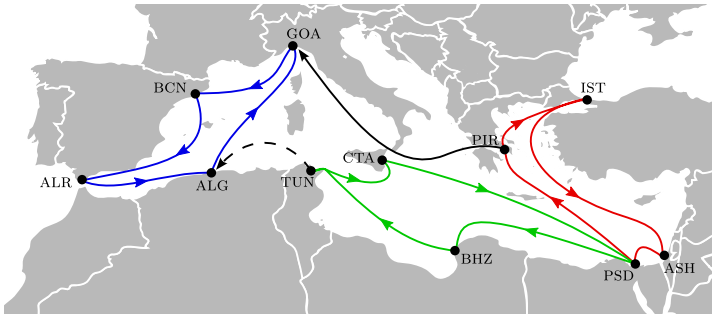


Figure 3.1: Vessels are repositioned from the red and green services to the blue service.

to two different ports on the blue service. Behind this picture lie the many details of repositioning: when and where the vessels should leave their initial services, what the vessels should do along the way to the goal service, and when and where should the vessels arrive at their goal service?

What makes repositioning problems particularly interesting is that repositioned vessels have a number of decisions to make besides where to leave a service and where to join their goal service. Vessels may undertake a number of activities along the way to their destination, such as carrying empty containers (*equipment*), sailing on other services within the network (*sail-on-service*), and adding (*inducing*) / removing (*omitting*) port calls. All the while, the speed of the vessel can be optimized to either conserve fuel or reach its destination faster, and containers are loaded and unloaded to prevent the disruption of normal cargo flows.

There is considerable room for shipping lines to save money while repositioning. Cleverly planned repositionings can result not only in lower costs for shippers, but also profits in certain situations in which customer demands can be satisfied. Repositioning is therefore a critical problem in the operations of shipping lines, and one that offers great potential for cost savings through the use of automated techniques.

The goal of repositioning problems is to determine the minimal cost paths from the vessels' initial services to the goal service, such that cargo flows are minimally disrupted. Avoiding cargo flow disruptions can be seen as a problem of maximizing revenue in a subset of the network, thus, we will look at repositioning problems both as a cost minimization problem, and as one of profit maximization.

Repositioning problems concern themselves with a subset of the overall network that is being changed, which stands in sharp contrast to other liner shipping problems, such as network design (e.g. [3, 17]). This is due to the fact that the repositioning of a group of ships only has an impact only on those ships' initial services, the goal service, and services with specific trade structures identified by the repositioning coordinator as potential sail-on-service opportunities. This helps keep the size of repositioning problems manageable, but it by no means indicates that repositioning problems are easy. On the contrary, the many activities available to vessels during repositionings result in many different possibilities that solution methods must sift through to find

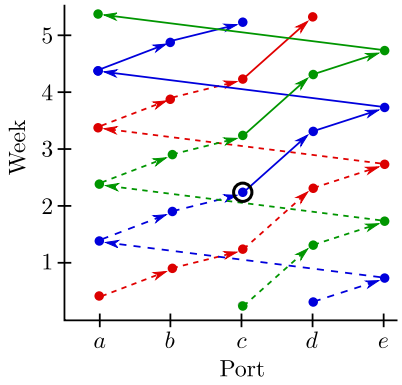


Figure 3.2: A time-space graph of a service with three vessels, with a latest phase-in requirement of port c in week 3.

the one (or ones) with the optimal cost.

Fleet repositioning is an NP-hard problem, which we show later in Chapter 6 through a reduction to the knapsack problem.

3.2 Phase-out & Phase-in

The repositioning period for each vessel starts at a specific time when the vessel may cease normal operations, that is, it may stop sailing to scheduled visitations and go somewhere else. Each vessel is assigned a different time when it may begin its repositioning, or *phase-out* time. After this time, the vessel may undertake a number of activities to lower the repositioning costs or earn revenue. Vessels may stay on their initial service past the phase-out time if it is economically viable to do so.

In order to complete the repositioning, each vessel must *phase in* to a slot on the goal service before a time set by the repositioning coordinator. After this time, normal operations on the goal service are set to begin, and all scheduled calls on the service are to be undertaken. In other words, the repositioning of each vessel and the optimization of its activities takes place in the period between two fixed times, the vessel's earliest phase-out time and the latest phase-in time of all vessels. Before the phase-in time of a service, vessels may begin regular operations on the service, if it is profitable to do so. The overall goal during this time period is to maximize the revenue earned by the repositioning vessel, which involves minimizing transportation costs and maximizing container transports.

Consider Figure 3.2, which shows a service with a phase-in deadline at port c in week 2 (circled). The solid lines connect all the visitations that must be undertaken, whereas the dashed lines connect visitations that will only be carried out if they are profitable during the repositioning.

Fixed costs

Between the phase-out and the phase-in, vessels pay a fixed hourly cost, in shipping parlance the *hotel cost*, just for being in an operational state. This cost can be included in fleet repositioning models or ignored, depending on whether coordinators want to optimize the usage of vessels. The hotel cost manifests itself mainly in fuel costs, as the cost of crews and vessels are considered a fixed cost of the shipping line that cannot be optimized in an operational problem like repositioning. Note that the hotel cost refers only to non-sailing costs incurred. These costs include, for example, generating electricity for the ship, especially the reefer containers, and heat for the vessel. Vessels require several metric tons of fuel per day depending on their size.

Trade zones

Trade zones are contiguous geographical areas consisting of a number of ports with a common nationality or supra-nationality, such as the EU trade area. We use trade zones to model restrictions preventing cargo from being brought somewhere it might violate the law, or violate a customer's contract on where the shipment may travel. Within trade zones, we allow vessels to sail freely between ports. Trade zones are relevant for a number of repositioning activities, such as the calling of ports with cargo or equipment, as well as in cabotage restrictions that are present in certain repositioning activities.

3.3 Repositioning Activities

A number of activities are available to vessels as they travel from their initial services to the goal service that can lower the cost of repositioning or earn money for the shipping line.

3.3.1 Sailing & Slow-steaming

The core activity used by repositioning vessels is *sailing*, in which vessels travel between ports across the open ocean. Sailing consumes bunker fuel, the consumption of which increases cubically with the speed of the vessel (see Section 2.2.3 for more information). *Slow steaming*, in which vessels sail near or at their minimum speed allows vessels to sail cheaper between two ports than at higher speeds, albeit with a longer duration. Figure 3.3 shows the total cost of fuel for traveling between Boston, USA and Copenhagen, Denmark by sea with three different vessels. This figure ignores fixed costs, but even if they were to be included sailing slowly saves money. For a post-Panamax vessel (cyan circles), the cost of sailing can be lowered from nearly $1 million to around $300,000 by doubling the voyage time. Large ships benefit the most from slow steaming, but even the panamax vessel (red squares) and the feeder vessel (green triangles) see cost reductions from slow steaming.

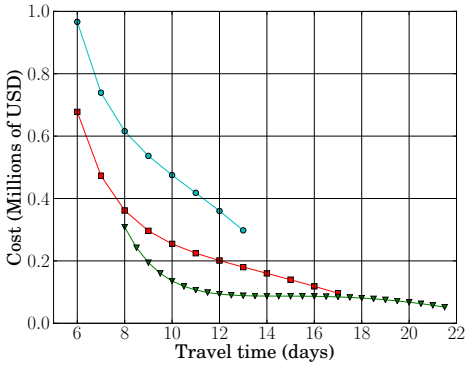


Figure 3.3: Travel cost (in millions of US dollars) from Boston, USA to Copenhagen, Denmark for a feeder vessel (green, triangles), Panamax vessel (red, squares), and a post-Panamax vessel (cyan, circles) with a bunker fuel price of 600 USD/mt. Data source: Anonymized data from Maersk Line.

Repositioning problems must therefore take vessel speed into account when optimizing vessel movements. Sometimes vessels must reach a tight phase-in deadline, and increasing costs is acceptable, whereas when deadlines are loose, vessels can sail slowly and reduce costs. However, slowing the speed of a vessel could reduce the amount of cargo that can be carried, as some customers may desire a faster delivery time than slow-steaming can provide.

3.3.2 Sail-on-service

While repositioning, vessels may use certain services designated by the repositioning coordinator to cheaply sail between two parts of the network. These are called *sail-on-service (SOS) opportunities*. There are two vessels involved in SOS opportunities, referred to as the *repositioning vessel*, which is the vessel under the control of a repositioning coordinator, and the *on-service vessel*, which is the vessel assigned to a slot on the service being offered as an SOS opportunity. Repositioning vessels use SOS opportunities by replacing the on-service vessel and sailing in its place for a portion of the service. We provide an example of an SOS which allows vessels to cheaply sail across the Pacific in our description of our case study in Section 3.4. SOS opportunities save significant amounts of money on bunker fuel, since one vessel is sailing where there would have otherwise been two. Using an SOS can even earn money from a *time-charter bonus*, which is money earned by the liner shipper if the on-service vessel is leased.

SOS opportunities often take place on the headhaul of a service (see Section 2.2.1 for more details), rather than on the backhaul. This offers a valuable opportunity to not only save money during repositioning, but also to save money for the shipper on the generally money losing backhaul. How this works is that the repositioning vessel travels on the headhaul of the service, eventually leaving the SOS service to continue its repositioning. Normally, a vessel would continue sailing on the backhaul, however,

since the repositioning vessel leaves the service to continue its repositioning, no vessel sails on the backhaul. Given the low amount of cargo present on the backhaul, this presents money saving opportunities for shipping lines.

When planning a repositioning, coordinators at shipping lines may designate particular slots of various services as SOS slots. The port calls of a particular SOS opportunity are divided into three contiguous sets, as designated by a repositioning coordinator: the start, middle and end. We now describe the three components of an SOS, as well as explain what happens to the on-service vessel once it is replaced by the repositioning vessel.

Starting and ending an SOS

The repositioning vessel takes over normal operations from the on-service vessel at one of the start ports. There are two ways in which the repositioning vessel may begin service on an SOS: *direct transshipment* or *parallel sailing*. In a direct transshipment, the repositioning vessel follows the on-service vessel to one of the start ports. All of the containers on the on-service vessel are then transshipped to the repositioning vessel. This can only be performed when such a transshipment will not violate a cabotage restriction. Furthermore, transshipments can be expensive, sometimes exceeding cost savings of an SOS. In such cases, a parallel sailing can be used for the repositioning vessel to join the service. The repositioning vessel joins the on-service vessel at one of the start ports, and the two vessels sail in tandem until either they reach the last start port or perform a direct transshipment at an earlier port. During the parallel sailing, the repositioning vessel loads cargo, and the on-service vessel discharges cargo. Although this means paying twice the bunker fees for the duration of the parallel sailing, this is sometimes still cheaper than transshipping large numbers of containers.

After joining the SOS, the vessel travels through a number of ports it is required to pass. These ports are designated by the repositioning coordinator as SOS middle ports. Vessels may not join or leave the SOS at this ports. This decision is made by the coordinator, and can be due to contractual obligations or various other business requirements. After the middle ports come the end ports, where the repositioning vessel may leave the SOS opportunity and sail elsewhere. It may leave from any one of the designated end ports.

Figure 3.4 shows an SOS opportunity and the various ways a repositioning vessel could utilize it. The service has three start ports, A, B, and C, however due to cabotage rules, B cannot be used for transshipment. The repositioning vessel may therefore perform a direct transshipment in ports A or C, or it may sail in parallel with the on-service vessel from A to C or from B to C. At C, the on-service vessel then leaves the service. The repositioning vessel stays on the service through the middle ports D and E. At the end ports F and G, the repositioning vessel may leave the service. A third vessel, shown in blue, joins the service at port A after the repositioning vessel has left. Note how there is a gap in service between either F and A or G and A, depending on where the repositioning vessel leaves. When this gap corresponds to the backhaul

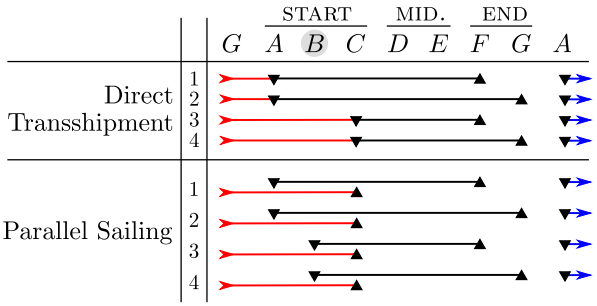


Figure 3.4: The various possibilities for starting and ending an SOS on a 7 port service (A through G), where the red line is the on-service vessel, the black line the repositioning vessel, and the blue line a third vessel continuing the service after the repositioning vessel leaves. Due to cabotage restrictions, a transshipment at port B is not allowed.

of a service, shipping lines can save significant amounts of money and avoid sailing a vessel empty.

On-service Vessel

When a repositioning vessel uses an SOS opportunity, the on-service vessel is displaced and must be handled by the repositioning coordinator. The coordinator has three main options. Vessels can be *laid up*, *leased out*, or repositioned to another service. Vessel lay ups involve removing a vessel from service, reducing the amount of crew on board, and mooring the vessel in or near a port for a period of time. Lay ups cost money, but can be cheaper than keeping a vessel in full operation when trade volumes decline. Some vessels can be leased out (time chartered) when they exit their service. Time charter rates vary, and vessels are not always in demand, especially with the large growth in the size of the world container ship fleet over the past several years [154]. It is up to the repositioning coordinator to decide whether the on-service vessel should be laid up, leased out, or itself repositioned. We consider the activities of the on-service vessel to be outside the scope of a single repositioning problem, and only include the on-service vessel insofar as its costs (or time charter profits) are accounted for.

3.3.3 Inducement & Omission

When a visitation is added to a service that is not part of the service's regular schedule, it is called an *inducement*. If a port on the initial or goal service is left off of the repositioning vessel's schedule, it is called an *omission*. Figure 3.5 shows a vessel's repositioning (solid blue line) from its initial service (dashed red) to its goal service (dotted green) within a trade zone. Although FXT is on both the goal and initial services, it is omitted from the repositioning. Note also that the ports RTM and BRV are induced onto the repositioning path.

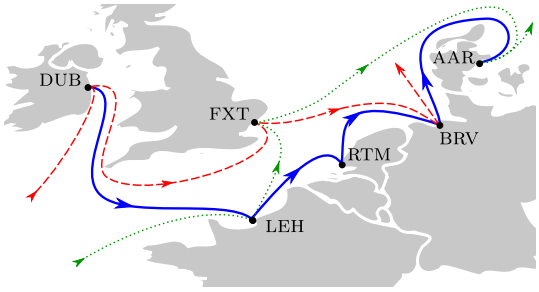


Figure 3.5: An example repositioning (blue) from a vessel's phase-out service (dashed red) to its phase-in service (dotted green).

3.3.4 Cargo Demands

Revenue is earned through delivering cargo. In any repositioning scenario, there are a number of *cargo demands* available, each of which has an origin visitation and a set of destination visitations. Note that there is a set of visitations, rather than a single destination, because the delivery visitation normally used may no longer exist in the scenario, for example if its service is closed down. We replace such deleted destinations with visitations at the same port with a time that is near the time the cargo was originally meant to be delivered. For example, consider a container that is supposed to be delivered as some port, p, on a particular Wednesday on the initial service. Due to a repositioning vessel phasing out, that visitation will no longer take place. However, perhaps the goal service of the vessel visits p on a Thursday, and an SOS opportunity visits p on a Monday. We can add those two visitations as destinations of the demand so that the containers are delivered. Delivering the demand to any of those destinations represents a satisfactory completion of the demand. Demands contain some number of TEU, and consist of either dry or reefer containers. Income is earned for each TEU of a demand delivered. That is, demands are allowed to be partially delivered.

Loading and unloading containers at ports costs money. The profit earned per TEU carried is therefore the revenue minus the loading and unloading costs. Vessels may carry any cargo demand available as long as they do not exceed their capacity in either the reefer or dry dimensions. Note that we only check whether the number of TEU loaded is less than or equal to the capacity of the vessel, and perform no detailed stowage planning (see, e.g., [113]) in order to check whether the vessel can sail or not. To the best of our knowledge, such an integration of vessel loading and liner shipping optimization, in terms of the fleet or network, has not been attempted.

The flow of cargo through the repositioning subset of a liner shipping network can be seen as a fractional multi-commodity flow. Cargo demands in fleet repositioning differ from, for example, pick-up-and-delivery problems (e.g., [42, 131]), in which all pickups and deliveries must be satisfied. In the LSFRP, partial demands may be carried and demands do not have to be satisfied if it would not be profitable.

3.3.5 Equipment

Trade imbalances result in empty container shortages and surpluses throughout a liner shipping network. Empty containers amass in ports receiving many shipments, and are often scarce in ports sending many shipments. Empty containers are important to have on hand, as customers are often provided empty containers to load as part of their contracts with a shipping line. Specifically, ports in North America and Europe tend to have empty container surpluses, whereas ports in South East Asia, especially China, generally have empty container deficits. In shipping parlance, empty containers are referred to as *equipment*, which we will use throughout this work.

In contrast to cargo, which has a specific origin and destination, equipment can be sent from any port where it is in surplus to any port where it is in demand. Each piece of equipment brought from a port where it is in excess to a port where it is needed earns a small revenue. The revenue earned is an estimation of how much money was saved by bringing the equipment on a repositioning vessel instead of moving the equipment through other, more expensive, means. Loading and unloading equipment is subject to a per TEU move cost that is either the same price or cheaper than loaded containers.

3.3.6 Flexible Visitations

Some ports have equipment or cargo, but are not on any service visited by repositioning vessels. Repositioning coordinators can try to schedule a repositioning vessel to go to those ports. We call these ports *flexible* ports, and they are associated with flexible visitations. All other visitations are called *inflexible*, because the time a vessel arrives is fixed. Flexible visitations tend to be at ports that are not overloaded with vessels, so acquiring a berth is not challenging. However, there is still no guarantee that the berthing time desired by the repositioning coordinator will be available, meaning the use of a flexible port comes with some risk that repositioning plans using such ports may not be feasible. Due to a lack of integrated systems between container terminals and shipping lines, an immediate check of berthing feasibility is not possible. This means that repositioning coordinators must make a repositioning plan, and then confirm with container terminals whether a berth is possible, and re-plan in the case that a berth cannot be secured.

3.4 Asia-CA3 Case Study

We performed a case study with our industrial collaborator, Maersk Line, to find out how they created new services in their network. We initially interviewed a number of Maersk Line employees. These employees include those responsible for the initial planning of services, employees who scheduled vessel berthing times with terminals, network planners who were concerned with the well-being of the overall network, employees managing the use of vessels, and the repositioning coordinators responsible for creating a

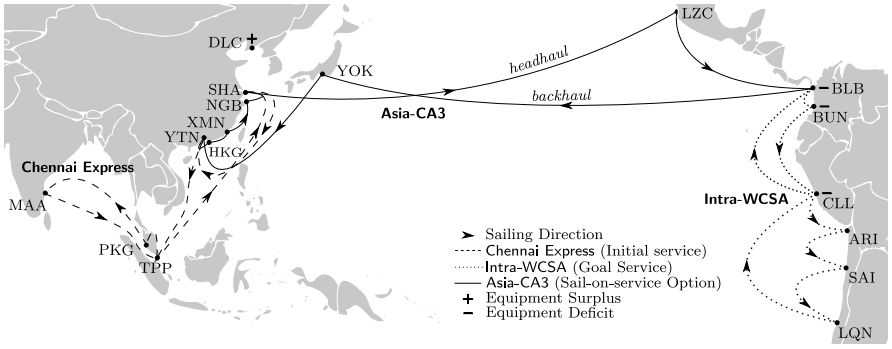


Figure 3.6: A subset of the case study we performed with our industrial collaborator.

repositioning plan. We followed up these initial interviews with more targeted interviews with repositioning coordinators to find out more about how repositioning actually works.

Our goal was to determine the decision making that went in to creating the Asia-CA3 service that brings containers from Asia, primarily China, to Central and South America. Over the course of this case study, we stumbled upon the problem of repositioning vessels, as the Asia-CA3 service was being used as an SOS opportunity during repositionings to the Intra-WCSA service in South America.

Figure 3.6 shows a subset of the overall Asia-CA3 case study. Several vessels are required to start the Intra-WCSA service, but these vessels are not available in South America. There are, however, vessels available for use on the Intra-WCSA in Asia. The Chennai Express service (left) is one of the several services from which vessels for the Intra-WCSA were sourced. The Asia-CA3 service is offered as a SOS opportunity to the vessel repositioning from Chennai Express to Intra-WCSA.

One example repositioning is for the repositioning vessel to leave the Chennai Express at TPP, and sail to HKG where it picks up the Asia-CA3, replacing the on-service vessel. The repositioning vessel then sails along the Asia-CA3 until it gets to BLB where it can join the Intra-WCSA. Note that no vessel sails on the backhaul of the Asia-CA3, and this is allowed because very little cargo travels on the Asia-CA3 towards Asia.

3.5 Related Problems

Fleet repositioning problems have received little attention in the literature and were not mentioned in any of the most influential surveys of work in the liner shipping domain [23, 25] or container terminals [137, 139]. We note that the latest review of shipping research [24] also makes no mention of repositioning, despite the publications of [147, 149] in 2012. The primary work on fleet repositioning is in [143, 144, 145, 147, 149]. These articles are the focus of this dissertation and will be covered in the following chapters.

3.5.1 Shipping Problems

There is a long history of research in the general shipping domain, which extends back to Ronen's well known reviews of ship scheduling and routing problems in the 70s and early 80s [125] and the 80s and early 90s [126]. We briefly consider problems from the liner shipping and tramp/industrial shipping literature and contrast them with fleet repositioning.

Liner Shipping

Although there has been significant work on problems such as the Fleet Deployment Problem (FDP) (e.g., [120]) and the Network Design Problem (NDP) (e.g. [3, 4, 17]), these problems deal with strategic decisions related to building the network and assigning vessels to services, rather than the operational problem of finding paths for vessels through the network. Specifically, the NDP focuses on constructing a liner shipping network given a set of ports and vessels. There are a number of different versions of the problem, but in general the network is constructed either to satisfy as much demand as possible (i.e., profit is maximized), or it is constructed to satisfy all demands at minimal cost. While there are some similarities between the NDP and fleet repositioning in that the NDP must build routes over which demands are then flowed in order to calculate the network profit (or cost), the NDP lacks many of the fixed time components of fleet repositioning, and the goal is to build a network, not move a vessel through a network. In the case of the FDP, vessels are assigned to routes in a pre-determined liner shipping network. The problem focuses on optimizing the operating costs of vessels on various services, rather than on the task of repositioning the vessels to those services.

The liner shipping vessel schedule recovery problem (VSRP) [18] focuses on recovering operations after a disruption, such as bad weather or mechanical failure, delays a container vessel. Similar to fleet repositioning, the VSRP must respect the weekly frequency of services and network cargo flows. However, the two problems differ in that the VSRP lacks many cost saving aspects of the LSFRP because it is solved over a much shorter time window.

Andersen's PhD thesis [5] discusses a fleet repositioning problem called the Network Transition Problem (NTP). No mathematical model or formal problem description is provided, so it is difficult to exactly ascertain what the NTP solves. However, it is clear that the NTP lacks cost saving activities like SOS opportunities, empty equipment flows and slow steaming.

Tramp Shipping

Although tramp shipping problems, such as [22, 89], maximize cargo profit while accounting for sailing costs and port fees as in fleet repositioning, they lack liner shipping specific constraints, such as sail-on-service opportunities, phase-in requirements and strict visitation times. In [138], the authors describe the maritime pickup and delivery

problem with time windows and split loads (MPDPTWSL), a ship routing and scheduling problem in which vessels must service some number of mandatory demands, and may carry other, optional demands if it would be profitable to do so. The primary difference between fleet repositioning and the MPDPTWSL is that the vessels in the MPDPTWSL are not required to sail to any particular locations the way they are in the LSFRP. In other words, the goal is solely to satisfy cargo demands, rather than to move the vessels to a particular location. Additionally, demands are handled quite differently between the two problems, as the LSFRP does not allow split deliveries, due to timeslots that are reserved for vessels to pick up and deliver containers.

3.5.2 Vehicle Routing Problems

Fleet repositioning shares some characteristics with the pickup and delivery problem (PDP) and the pickup and delivery problem with time windows (PDPTW) [42, 131], in that vehicles service some number of demands from customers. However, all demands must be satisfied in the PDP and PDPTW, whereas in fleet repositioning demands are only a way to increase profit. Equipment flows also represent a major difference between the two types of problems, in that they have a many-to-many flow structure not present in the PDPTW. The vehicle routing problem with split deliveries (SDVRP) (see [8]) differs even more so from fleet repositioning in that cargo emanates from the depot and is then brought to customers.

3.5.3 Airline Disruption Management

Airline disruption management (see [26, 87]), while also relying on time-based graphs, differs from fleet repositioning in two key ways. First, airline disruption management requires an exact cover of all flight legs over a planning horizon. Fleet repositioning has no such requirement over visitations or sailing legs. Second, there are no flexible visitations in airline disruption management.

3.6 Chapter Summary

In this chapter, we presented liner shipping fleet repositioning, an expensive task often faced by the world's shipping lines. We described the various activities container ships may undertake while repositioning to a new service, as well as the liner shipping specific restrictions that ships must obey. Despite this problem's practical importance, and the cost savings that could be achieved through the automated generation of repositioning plans, it has not yet been considered in the literature.

Chapter 4

Methodological Background

We provide background information on several key topics for understanding the models of fleet repositioning and inter-terminal transportation in this dissertation. Section 4.1 provides background on automated planning, a well-known technique for selecting and sequencing activities in order to achieve a set of goals. We then describe partial-order planning (POP), a specific type of automated planning used and extended in this dissertation, in Section 4.2. Next, we cover mixed-integer programming (MIP), a method for solving optimization problems with linear constraints and objectives in Section 4.3. Finally, we describe constraint programming (CP), a branch-and-bound technique for satisfaction and optimization problems that uses constraint propagation and backtracking search to solve combinatorial problems.

4.1 Automated Planning

Automated planning is used to model and solve problems where it is difficult to select and sequence activities in order to achieve specific goals starting from an initial state. In the case of the automated planning approaches used in this work, this sequence of activities is a subset of the overall space of activities. In general, this is not the case, and this allows automated planning to model PSPACE-complete problems [54].

We describe a state variable based version of propositional STRIPS planning [46]. In state variable planning, a state is represented by an assignment of a set of variables. There are a number of versions of state variable planning, several of which are undecidable [62]. We therefore base our descriptions on a version of state variable planning in which state variables receive an assignment from a finite domain, without allowing complex numerical effects. This version of state variable planning is decidable [62]. Although this usage of state variables is not more expressive than propositional planning, it allows for more intuitive modeling, as well as makes mutual exclusivity of variable values an explicit feature of a model. Activities are represented by actions that can only be applied in states that satisfy their precondition. When applied, actions change a state, thus reassigning some of the state variables representing the state.

Formally, let $\mathcal{V} = \{v_1, \ldots, v_n\}$ denote a set of *state variables* with finite domains $D(v_1), \ldots, D(v_n)$. A *state variable assignment* ω is a mapping of state variables to values $\{v_{i(1)} \mapsto d_{i(1)}, \ldots, v_{i(k)} \mapsto d_{i(k)}\}$ where $d_{i(1)} \in D(v_{i(1)}), \ldots, d_{i(k)} \in D(v_{i(k)})$. We also define $vars(\omega)$ as the set of state variables used in ω.

We define a *state variable planning problem* to be a tuple $\langle \mathcal{V}, \mathcal{D}, \mathcal{A}, \mathcal{I}, \mathcal{G}, pre, \mathit{eff} \rangle$, where $\mathcal{D}$ is the Cartesian product of the domains $D(v_1) \times \cdots \times D(v_n)$, $\mathcal{A}$ is a set of actions, $\mathcal{I}$ is a total state variable assignment representing the initial state (i.e., $vars(\mathcal{I}) = \mathcal{V}$), $\mathcal{G}$ is a partial assignment representing the goal states (i.e., $vars(\mathcal{G}) \subseteq \mathcal{V}$), pre_a is a partial assignment representing the precondition of action a, and eff_a is a partial assignment representing the effect of action a.[1]

Let $S = \{\omega \mid vars(\omega) = \mathcal{V}\}$ denote the set of all the possible states. An action a is applicable in $s \in S$ if $pre_a \subseteq s$ is true and, if applied, is assumed to cause an instantaneous transition to a successor state defined by the total assignment

$$succ_{a,s}(v) = \begin{cases} \mathit{eff}_a(v) & \text{if } v \in vars(\mathit{eff}_a), \\ s(v) & \text{otherwise.} \end{cases} \tag{4.1}$$

A *plan* is represented by a pair $\langle A, O \rangle$, where $A \subseteq \mathcal{A}$ is the set of actions in the plan[2] and O is a set of ordering constraints of the form $a \prec b$, where $a, b \in A$. A plan is a solution to a state variable planning problem iff for any total order of the actions in the plan $a_1 \prec a_2 \prec \cdots \prec a_{|A|}$ that is consistent with O, there exists a state sequence $s_1 \ldots s_{|A|}$ such that $s_1 = \mathcal{I}$; $s_{|A|} \supseteq \mathcal{G}$; and for $1 < i \leq |A|$, $pre_{a_i}(s_{i-1})$ is true and $s_i(v) = succ_{a_i, s_{i-1}}(v), \forall v \in \mathcal{V}$.

Numerous planners exist for solving automated planning problems. For an overview we refer to the reviews of several of the international planning competitions, [27, 52, 96] in which planners were systematically compared against each other on a diverse dataset of planning instances.

4.2 Partial-Order Planning

A well-known method for solving planning problems is *partial-order planning* (POP) [117]. POP has a long history in the planning community, but has been recently eclipsed by the strong performance of heuristic search based forward planners. Nonetheless, partial-order style planners are still relevant within the planning community [107, 166], and their ideas have been integrated into forward planning approaches [33, 92].

[1] In practice, is it often more convenient to represent actions in a more expressive form, e.g., by letting the precondition be a general expression on states $pre_a : S \to \mathbb{B}$ and represent conditional effects like resource consumption by letting the effect be a general transition function, depending on the current state of S, $\mathit{eff}_{a,s} : S \to S$. Such expressive implicit action representations may also be a computational advantage. We have chosen a ground explicit representation of actions because it simplifies the presentation and more expressive forms can be translated into it.

[2] We emphasize again that in automated planning, actions can actually appear multiple times in a plan. However, for the purposes of this thesis, restricting plan actions to be a subset of all actions is sufficient. We refer to [54] for an overview of general automated planning.

Starting from an empty plan, actions are bound together by *causal links*, in which the effect of one action fulfills a precondition of another action. As its name implies, POP imposes only a partial-order over the ordering of actions chosen for the plan, rather than a total order as in forward planning. In forward planning, such actions must be ordered, even when the ordering is superfluous. Consider, for example, two ships with sailing actions. The actions are completely non-interacting, and in POP these actions would not require an ordering. Actions that use a shared resource, such as a crane picking up a container, however, require an ordering, as two cranes cannot pick up the same container at the same time. POP methods detect these situations and post ordering constraints between actions to ensure the feasibility of a plan. Partial-order planners also check for conflicts between actions that would prevent them from executing simultaneously, and post the appropriate ordering constraints.

We use the same problem description for POP as in Section 4.1, and build off of the definition of a plan. A POP plan is represented by a tuple $\pi = \langle A, O, C \rangle$, where A and O are a set of actions and orderings as previously defined and C is a set of causal links. A causal link $a \xrightarrow{\mu} b$, where $a, b \in A$; $\mu = v \mapsto d$ is a single state variable assignment with $v \in \mathcal{V}$, $d \in D(v)$; $\mu \in \mathit{eff}_a$; and $\mu \in \mathit{pre}_b$, represents the fulfillment of action b's precondition μ by the effect of action a. Causal links connect an effect of one action to a precondition of another action, satisfying that particular precondition. Each causal link is associated with a single state variable mapping. This state variable mapping is contained in the effect of the action linked from, and is also in the precondition of the action being linked to. Using causal links allow us to build a representation of a plan that is not totally ordered. We adopt the same model of state succession in partial-order planning as in equation (4.1).

Every plan contains at least two actions, a_0 and a_∞. The action a_0 represents the initial state of the problem, $\mathcal{I}$, with $\mathit{pre}_{a_0} = \emptyset$ and $\mathit{eff}_{a_0} = \mathcal{I}$. The action a_∞ represents the goal state, $\mathcal{G}$, with $\mathit{pre}_{a_\infty} = \mathcal{G}$ and $\mathit{eff}_{a_\infty} = \emptyset$.

Flaws

In POP, deficiencies in a plan are called *flaws*. These flaws must be repaired in order to find a complete plan that satisfies the goal state.

An *open condition* $\xrightarrow{\mu} b$ is an unfulfilled precondition μ of action $b \in A$, that is, $\mu \in \mathit{pre}_b$ and $\forall a \in A$, $a \xrightarrow{\mu} b \notin C$. Open conditions indicate that a particular precondition does not yet have support from an effect of another action or the initial state. The addition of a causal link from an open condition to an action with a satisfying effect removes the open condition. Note that the addition of a new action to the plan may be necessary in order to satisfy an open condition. Formally, an open condition flaw $\xrightarrow{\mu} b$ can be repaired by linking μ to an action a such that $\mu \in \mathit{eff}_a$ and by posting an ordering constraint over a and b. Thus, $C \leftarrow C \cup \{a \xrightarrow{\mu} b\}$ and $O \leftarrow O \cup \{a \prec b\}$. In the case that $a \notin A$, $A \leftarrow A \cup \{a\}$ and $O \leftarrow O \cup \{a_0 \prec a, a \prec a_\infty\}$.

An *unsafe link* (also known as a *threat*) is a causal link $a \xrightarrow{\mu} b$ that is threatened by an action c such that *i)* $\mathit{vars}(\mu) \in \mathit{vars}(\mathit{eff}_c)$, *ii)* $\mu \notin \mathit{eff}_c$, and *iii)* $\{a \prec c \prec b\} \cup O$ is

Algorithm 4.1 Partial-order planning algorithm, based off of [164] and [117].

```
1: function POP(I, G)
2:     Π ← {InitialPlan(I,G)}
3:     while Π ≠ ∅ do
4:         π ← SelectPlan(Π)
5:         Π ← Π \ {π}
6:         if NumFlaws(π) = 0 then
7:             return π
8:         else
9:             f ← SelectFlaw(π)
10:            Π ← Π ∪ RepairFlaw(π, f)
11:    return "Infeasible problem"
```

consistent. Causal links are threatened by actions that have an effect that changes their associated state variable, if that action can be ordered between the actions connected to the causal link. An unsafe link $a \xrightarrow{\mu} b$ that is threatened by action c can be repaired by either adding the ordering constraint $c \prec a$ (*demotion*) or $b \prec c$ (*promotion*) to O.

Together, open conditions and unsafe links constitute the flaws in a plan. Let $\mathit{flaws}(\pi) = \mathit{open}(\pi) \cup \mathit{unsafe}(\pi)$ be the set of flaws in the plan π, where $\mathit{open}(\pi)$ is the set of open conditions and $\mathit{unsafe}(\pi)$ is the set of unsafe links. We say that π is a *complete plan* if $|\mathit{flaws}(\pi)| = 0$, otherwise π is a *partial plan*. In other words, a plan is considered complete when all of the preconditions of all of its actions, including the goal state, are satisfied through a causal link to another action or to the initial state, and there are no unsafe links.

Searching for a Plan

POP solvers search through the space of partial plans, starting with a plan containing the actions a_0 and a_∞, as previously described. POP solvers then iteratively eliminate flaws in the plan, by adding actions, causal links, or ordering constraints, until all of the preconditions of the actions in the plan have a causal link connecting them to other actions, or they prove that no satisfying plan exists.

Algorithm 4.1 shows a generic method for solving a POP problem, based off of [164] and [117]. The algorithm is initialized with the initial state, $\mathcal{I}$, and goal state, $\mathcal{G}$. A set of partial plans, Π, is initialized with a plan π_{init} from the InitialPlan function (line 2). Let $\pi_{init} = \langle \{a_0, a_\infty\}, \emptyset, \{a_0 \prec a_\infty\} \rangle$, where a_0 and a_∞ are as previously described.

The algorithm then selects a plan from Π (line 4) and checks the number of flaws in the plan (line 6). If the selected plan has no flaws, it is a complete plan and is therefore returned, otherwise a flaw is selected and repaired (lines 9 and 10). This process repeats until either a complete plan is found or Π is empty, in which case the POP problem is infeasible.

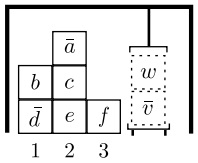


Figure 4.1: In Example 4.1, a container crane loads containers onto a double-stack railroad car. The car carries two containers, a refrigerated container in slot $\bar{v}$ and a regular container in w. Refrigerated containers are marked $\bar{a}$ and $\bar{d}$, and b, c, e and f are regular containers.

SelectPlan(Π) returns a single plan from Π and SelectFlaw(π) returns a flaw from π. These are generic procedures that could be based on any number of plan and flaw selection heuristics in the literature for POP (e.g. [61, 99, 166]).

RepairFlaw(π, f) returns a set of plans representing all of the possible ways of fixing the flaw f in π, unless f cannot be fixed, in which case the empty set is returned. Under our definitions of SelectPlan, SelectFlaw and RepairFlaw, Algorithm 4.1 is a complete algorithm. That is, the algorithm will always return a complete plan if there is one, and it will always return "Infeasible problem" if no complete plan is possible to produce. The algorithm can be very easily modified to be a heuristic approach to planning by changing the RepairFlaw function to only return a subset of the possible partial plans for fixing a flaw. In order to illustrate partial-order planning, we construct the following example based on a toy problem involving the loading of a railroad car from several stacks of containers.

Example 4.1. In the *container loading problem*, the operator of a crane must load containers onto a double-stack railroad car as shown in Figure 4.1. Slot $\bar{v}$ must carry a refrigerated container, and slot w may not carry a refrigerated container. No container may be placed in slot w until $\bar{v}$ is full and containers can only be moved if they are at the top of a stack. Finally, containers can only be moved from their stack to the car if there are no tall container stacks blocking the path.

The task is to create a plan for the crane operator to fully load the railroad car. We add the additional requirement that container b must be loaded into slot w. Figure 4.2 shows a partial plan for this problem, where the state variables $occ_s \in \mathbb{B}$ represents whether or not slot s is occupied by a container, $top_t \in \mathbb{B}$ denotes whether container t is on the top of a stack, and $in_s \in t$ describes which container t is in slot s on the rail car, with $s \in \{w, \bar{v}\}$ and $t \in \{\bar{a}, b, c, \bar{d}, e, f\}$. There is a single ungrounded action in the planning domain, $move(t, s)$, which moves container t from its location in the stacks onto the rail car in slot s.

In the partial plan in Figure 4.2, an ordering constraint ensures that $move(\bar{a}, \bar{v})$ comes before $move(b, w)$. This constraint is posted to the model in response to the threat that the effect occ_w of $move(b, w)$ poses to the causal link between $\mathcal{I}$ and $move(\bar{a}, \bar{v})$, which requires $\neg occ_w$.

Notice also that due to the height of stack 2, container b can only be loaded onto the rail car if container $\bar{a}$ is moved out of the way first. This will result in a causal link

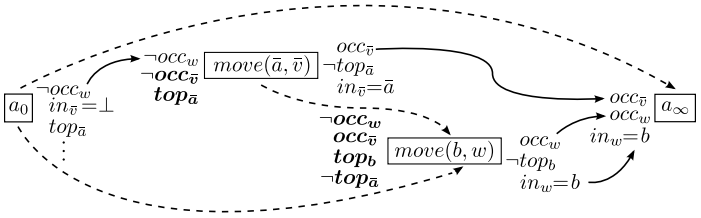


Figure 4.2: A partial plan for the container loading problem. Actions are shown in boxes with preconditions to the left and effects to the right. Actions are connected by causal links (*solid lines*) and ordering constraints (*dashed lines*), and open conditions are shown in bold.

between the actions later in the plan, when the open condition $\neg top_a$ of $move(b, w)$ is fulfilled by the effect $\neg top_a$ of $move(\bar{a}, \bar{v})$.

4.3 Linear and Mixed-Integer Programming

Mixed-integer programs (MIPs)[3] are optimization problems of the form:

$$\begin{aligned} &\text{minimize } \mathbf{c}^\intercal \mathbf{x} \\ &\text{subject to } A\mathbf{x} \geq \mathbf{b} \\ &\qquad \mathbf{x} \geq 0 \\ &\qquad x_i \in \mathbb{Z}, \forall i \in \mathcal{I}, \\ &\qquad x_i \in \mathbb{R}, \forall i \notin \mathcal{I}, \end{aligned}$$

in which $\mathbf{x}$ is a vector of n decision variables, $\mathbf{c} \in \mathbb{R}^n$ is a vector of objective coefficients, $A \in \mathbb{R}^{m \times n}$ is an $m \times n$ matrix of constraint coefficients, $\mathbf{b} \in \mathbb{R}^m$ is a vector of upper bounds on constraints, and $\mathcal{I} \subseteq \{1, \ldots, n\}$ indicates which decision variables take an integer value. When $\mathcal{I} = \{1, \ldots, n\}$, i.e., all of the variables take integer values, the problem is known as an *integer program* (IP). When $\mathcal{I} = \emptyset$, the problem is a *linear program* (LP). Unlike integer and mixed-integer programs, which are NP-hard to solve [77], linear programs can be solved in polynomial time [84]. Nonetheless, the simplex algorithm, which has an exponential worst-case runtime, is often used [115] due to its good performance for solving LPs, which often even beats algorithms with a worst-case polynomial runtime complexity.

Solving MIPs

MIPs are generally solved using a *branch-and-bound* approach in which all of the possible settings of the decision variables $\mathbf{x}$ are enumerated through a systematic tree search.

[3] In this work, we only refer to mixed-integer linear programs.

A lower bound can be found for each tree node using its *LP relaxation*. The LP relaxation is computed by modifying the original MIP such that the integer restrictions on variables are relaxed. That is, all variables are given a continuous domain. This relaxation can then be efficiently solved in an LP solver. The LP relaxation is then used to determine whether a branch is dominated and should be pruned. Additionally, the LP relaxation can be used in heuristics that decide which branches to explore first during the search.

Another approach is *branch-and-cut*, which successively adds *cuts*, which are linear constraints, to a problem. Branch-and-bound can be seen as a form of branch-and-cut in which branching is performed by adding cuts to impose domain restrictions on variables. At the root node of the search, branch-and-cut adds cuts to attempt to constrain the feasible region of the problem such that the solution of the LP relaxation yields an integer solution, which would be the optimal solution to the problem. Many cuts are problem specific and target special structures in problems such as network flows, knapsack constraints or set covering constraints (see, e.g., [98]). More general cuts also exist, such as the well-known Gomory cuts [55]. Should this cut generation procedure fail to solve the problem, branching (through cuts) is then performed.

Solution techniques for MIPs are well studied, and there are numerous algorithms and heuristics for solving MIPs. A number of commercially available solvers exist to solve MIPs, such as IBM CPLEX [69], SCIP [2], and Gurobi [58].

Column Generation

As the number of variables grows, LPs become increasingly difficult to solve. In some problem formulations there can even be an exponential number of variables. Standard approaches to solving linear programs are insufficient for these problems, as the problem with all of its variables often cannot even be loaded into memory. *Column generation* (also called *delayed column generation*) is a technique for solving such problems in which only a subset of all of the columns, i.e., variables, of a problem are considered at any single time, and variables are added as needed to solve the problem.

Column generation is used to solve a number of optimization problems, including vehicle routing with time windows [75] and ship scheduling [133]. We note that since column generation is used to solve LPs, it cannot provide solutions (in general) to MIPs. However, column generation can be used to find a lower bound that a MIP can then use as part of its solution procedure.

We briefly describe column generation based on the description in [39] and adopt its notation. We refer to [39] for a more general overview of column generation. In column generation, we use a decomposition of the problem into a *master problem* and *subproblem* in order to handle the large number of columns. Columns are iteratively added to a *restricted master problem* (RMP) which only contains a subset of the columns in the master problem. The RMP starts out only with the columns necessary to define a starting feasible solution (or an artificial solution), and then relies on a subproblem to determine which columns should be added at each iteration. The algorithm is finished

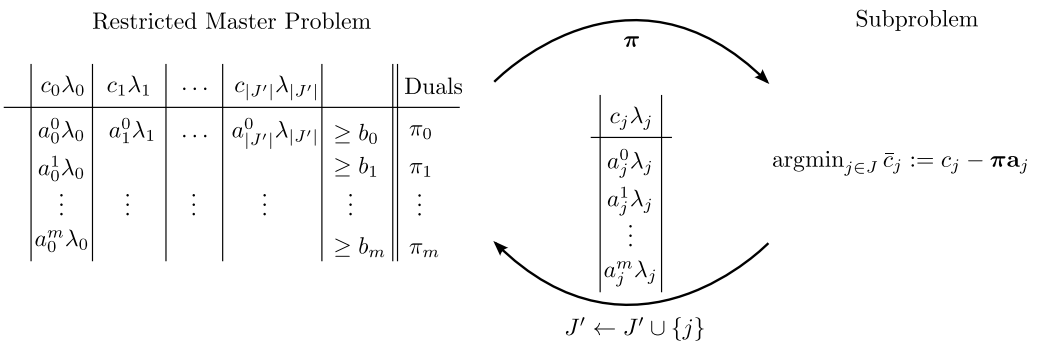


Figure 4.3: A visual overview of the column generation procedure in which a single column is generated from the subproblem.

when there are no columns left for the subproblem to add that can improve the solution, a notion that we will define formally.

The master problem takes the form:

$$\begin{aligned} \text{minimize} \quad & \sum_{j\in J} c_j\lambda_j \\ \text{subject to} \quad & \sum_{j\in J} \mathbf{a}_j\lambda_j \geq \mathbf{b} \\ & \lambda_j \geq 0, \forall j \in J, \end{aligned}$$

where J is the set of columns, $\mathbf{a}_j$ is a column in the $|J| \times m$ matrix A, with m as the number of constraints, λ_j is the decision variable corresponding to column j, and $\mathbf{b}$ is a vector of constraint coefficients. We associate the dual variables $\boldsymbol{\pi}$ with the constraints (i.e., $\boldsymbol{\pi}$ is a length m vector). The RMP is solved over a subset $J' \subseteq J$, as $|J|$ tends to be exponentially large in the size of the problem input.

In each iteration of column generation, the goal is to find a $j \in J$ minimizing the reduced cost $\bar{c}_j := c_j - \boldsymbol{\pi}^\intercal\mathbf{a}_j$. Since enumerating every column j would take too long, columns are generated by solving a pricing problem.

Figure 4.3 shows the column generation procedure in which columns are generated in a subproblem using information gained from the dual variables in the restricted master problem. Given $\boldsymbol{\pi}$, the dual optimal solution of the RMP, we can compute $\bar{c}^* := \min\{c_j - \boldsymbol{\pi}^\intercal\mathbf{a}_j \mid j \in J'\}$, where $J' \subseteq J$ is the set of columns currently not in the RMP. When $\bar{c}^* \geq 0$, there are no more columns to generate, and the solution to the RMP is optimal. Otherwise, a column with $\bar{c}_j < 0$ is added to the RMP and it is re-solved. Note that multiple columns can be added to the RMP at a single time, but from a theoretical perspective, this is not necessary. Additionally, the column with the minimal $\bar{c}_j$ value is not required to be added; any column with a negative $\bar{c}_j$ can be appended to the RMP. This could be done through a heuristic procedure. However, in order to prove the optimality of an RMP solution $\boldsymbol{\lambda}$, it must be shown that no column exists with a $\bar{c}_j < 0$.

In many cases, the master problem takes the form of a set partitioning, covering or packing problem and the subproblem is a shortest path problem with side constraints or a knapsack problem. Algorithms specifically designed for these problems can be used in both the master and subproblem in order to speed up solution time. For more information, see, e.g., [103].

4.4 Constraint Programming

Constraint programming (CP) is a declarative modeling paradigm in which the relations between variables are specified through a set of constraints [128]. In contrast to mixed-integer programming, which only deals with linear objectives and constraints, CP is able to handle general (i.e., non-linear) objectives and constraints. CP is used to solve two types of problems: *constraint satisfaction problems* (CSPs) and *constraint optimization problems* (COPs) (See [129]). The goal of a CSP is to find a satisfying assignment of values to a set of variables under some constraints, whereas in a COP, the goal is to find a satisfying assignment with an optimal (i.e., minimal or maximal) objective value satisfying the constraints.

4.4.1 CSPs and COPs

Formally, a CSP is defined by a tuple $\langle X, D, C\rangle$, where X is a set of variables $\{x_1, \ldots, x_n\}$, D is the set of domains $\{D_1, \ldots, D_n\}$ such that $x_i \in D_i$, and C is a set of m constraints $\{C_1, \ldots, C_m\}$ where $C_j : D_1 \times \cdots \times D_n \to \mathbb{B}$ defines which settings of values are allowed by C_j. The goal of a CSP is to find an assignment of variables $\{v_1, \ldots, v_n\}$, such that $v_i \in D_i$, and $\bigwedge_{1<j\leq m} C_j(v_i)$ is true [129].

A COP is defined similarly to a CSP using the tuple $\langle X, D, C, f\rangle$, where X, D and C are as previously defined, and $f : D_1 \times \cdots \times D_n \to \mathbb{R}$ is the objective function. COPs can be solved through the iterated solving of CSPs where, after each solution s to a CSP is found, a constraint is posted to C that forces the objective function of the next solution to be less than the current solution [129]. When no further solution can be found, the optimal solution is known.

Comparing CSPs and COPs to MIPs, we see that COPs handle a much broader variety of objective functions and constraints. While MIPs require the objective and constraints to be linear combinations of the variables, both CSPs and COPs allow non-linear combinations of variables in the objective and constraints.

4.4.2 Solving CP Problems

CSPs and COPs are solved using CP, which involves a backtracking search procedure combined with *constraint propagation* that performs deep reasoning to remove infeasible values from the domains of variables. CP allows *global constraints*, which are

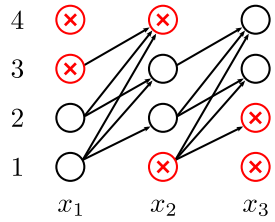


Figure 4.4: Arc consistency graph for Example 4.2.

constraints involving multiple variables that provide specialized domain filtering algorithms to exploit the structure present in various types of constraints.

Search, Propagation and Filtering

The backtracking search procedure for CSPs and COPs is similar to the branch-and-cut procedure used for solving MIPs not only in terms of its exploration of a search tree, but also in that domain pruning is used to try to speed up the search. In the case of branch-and-cut, this is done with the strategic addition of cuts. In CP, inferencing techniques and constraint propagation are used at each search node to forbid particular variable-value assignments [12].

Example 4.2. Consider an example CSP consisting of the variables x_1, x_2 and x_3, all with the domain $\{1, 2, 3, 4\}$, and the constraints $c_1 \equiv x_1 < x_2$ and $c_2 \equiv x_2 < x_3$.

Propagating c_1 in the example results in the value 1 being removed the domain of x_2, as well as the value 4 being removed from the domain of x_1. Propagating constraint c_2 removes values 1 and 2 from the domain of x_3, as well as the value 4 from the domain of x_2. After this single iteration of constraint propagation, the domains are thus $\{1, 2, 3\}$, $\{2, 3\}$, and $\{3, 4\}$ for x_1, x_2 and x_3, respectively.

Note that this form of propagation, in which each constraint is analyzed individually, is not strong enough to detect that x_1 will never take the value 3. All constraints can be taken simultaneously into account with *arc consistency*, which builds a graph using the support of the values of each variable based on the various constraints. Figure 4.4 shows the arc consistency network for the previous example. Circles represent variable-value assignments, and red circles with an "x" indicate assignments banned by arc-consistency. Arcs are drawn between variable-value pairs if the constraints allow both of the variables to be set to their associated values simultaneously. Using arc consistency, the variable x_1's domain is completely pruned, unlike when the constraints are propagated individually. Although there is support for the assignment $x_1 = 3$ in the domain of x_2 ($x_2 = 4$), this variable-value assignment has no support in the domain of x_3. Thus, it is not sufficient as the sole support of $x_1 = 3$, and thus 3 can be safely pruned from x_1's domain.

Numerous algorithms exist to enforce arc consistency that provide various trade-offs based on efficiency and memory usage, and we therefore refer to [12] for an overview. Other forms of consistency also exist, and we again refer to [12] for details.

Constraint propagation can be computationally expensive, meaning it is not always worthwhile propagating every constraint or completely pruning the domain of every variable. Partial invocations of constraint propagation do not affect the completeness of CP methods, but this can affect the efficiency of the overall search. On many real-world problems solved with CP, a compromise must be found that ensures enough pruning is performed to prevent unnecessary branching, but not so much that the search spends all of its time propagating constraints.

Should the domain of any variable becoming empty during constraint propagation, it is immediately clear that the CSP or COP is infeasible under the current constraints. If this occurs at the root node of the search, then the entire problem is infeasible. If this occurs at any other node in the search tree, further branching at that node is not required and the search backtracks up a level.

Once constraint propagation is unable to prune any values from any domain of any variable, or the propagation procedure is prematurely ended, and the solution can not be immediately deduced from the domains of the variables, branching must be performed. A number of branching strategies exist for CP, such as enumerating the values in the domain of a variable in each branch; binary choice point branching, in which a branch is generated setting a variable to a value in its domain, and a branch is generated preventing the variable from being assigned to that value; and domain splitting, in which a branch is created requiring a variable to be less than or equal to a particular value, and a branch requiring the variable to be greater than that value. We refer to [156] for a thorough overview of branching strategies and other considerations within backtracking search.

Global Constraints

Global constraints allow multiple variables to be simultaneously constrained in order to capture problem-specific structures. Maintaining such structures in the model tends to make problems easier to solve. Global constraints are coupled with *filtering algorithms* that perform constraint propagation for the constraint. This allows problem modelers to either use existing algorithms from the literature, or write their own algorithms to help solve their problem faster. We note that this solving procedure differs greatly from that used in the solving of MIPs, in that additional constraints or variables need not be generated. Rather, the variable domains are directly pruned by the filtering algorithm.

One of the most well-known global constraints is the `alldifferent`(X) constraint. The constraint accepts a set of variables, X, and enforces that every variable in X takes a different value. This constraint can also be represented in a pairwise (i.e., non-global) fashion for n variables, as follows: $x_i \neq x_j, \forall 1 < i < j \leq n$. However, the `alldifferent`$(X)$ constraint is able to exploit the fact that it is imposing the not equals relation over a number of variables in the way it filters the domains of constraints, providing an advantage to models that utilize the global `alldifferent`(X) constraint over a pairwise version. Numerous global constraints have been formulated in the literature, and we refer the reader to [158] for a complete overview.

				1	6	2	5	
	3		4					6
6	2	1			3			
	4	8	9		2		3	
7		1			8			5
	6		5		7	9	8	
			6			8	2	4
9					5		6	
	8	6	3	7				

Figure 4.5: A sudoku puzzle generated by `www.websudoku.com`.

Example 4.3. The `alldifferent` constraint can be used to model the well-known sudoku puzzle game [134]. Sudoku consists of a partially filled 9 by 9 matrix with 9 different 3 by 3 submatrices, as shown in Figure 4.5. Each submatrix must be filled in with the numbers 1 through 9. Each row and each column must also contain the numbers 1 through 9. We note that sudoku can be generalized to an n by n matrix, as long as $\sqrt{n}$ is an integer, and the CP model we present can be trivially extended to the general case. The following model (based off of the ideas in [134]) uses the variables $x_{ij} \in \{1, \ldots, 9\}$ for all $1 \leq i, j \leq 9$.

$$\texttt{alldifferent}(x_{1,j}, x_{2,j}, \ldots, x_{9,j}) \qquad \forall\, 1 \leq j \leq 9 \qquad (4.2)$$

$$\texttt{alldifferent}(x_{i,1}, x_{i,2}, \ldots, x_{i,9}) \qquad \forall\, 1 \leq i \leq 9 \qquad (4.3)$$

$$\texttt{alldifferent} \begin{pmatrix} x_{3i+1,3j+1}, & x_{3i+2,3j+1}, & x_{3i+3,3j+1}, \\ x_{3i+1,3j+2}, & x_{3i+2,3j+2}, & x_{3i+3,3j+2}, \\ x_{3i+1,3j+3}, & x_{3i+2,3j+3}, & x_{3i+3,3j+3} \end{pmatrix} \qquad \forall\, 0 \leq i, j \leq 2 \qquad (4.4)$$

Constraints 4.2 (constraints 4.3) ensure that each element in a row (column) has a different value. Constraints 4.4 prevents any two elements of any of the nine 3 by 3 squares from taking the same value. The advantage of this model over a model using pairwise constraints between variables is due to the strong filtering algorithms that have been published for the `alldifferent` constraint [159]. Furthermore, such constraints make the model easier to work with and interpret.

Chapter 5

Liner Shipping Fleet Repositioning without Cargo

In this chapter, we present the work on solving liner shipping fleet repositioning problems without cargo demands in [83, 146, 147]. These works focus on a simplification of the overall fleet repositioning problem, in which cargo demands are not taken into account, and the phase-out and phase-in times are optimized over a multiple week period. We call this problem the *no cargo liner shipping fleet repositioning problem* (NCLSFRP). Specifically, we include the following components of fleet repositioning in our model of the NCLSFRP:

1. Individual vessels' phase-out time and port are selected by the model over a multi-week period.
2. The latest phase-in time is chosen over a multi-week period.
3. The phase-in must result in a liner shipping service with weekly visitation at all ports on the service.
4. A single phase-in port is chosen for all vessels.
5. SOS opportunities must be started with a direct transshipment.
6. Equipment opportunities are represented as reduced sailing costs between ports with equipment surplus and ports with deficits.
7. Vessels' sailing speeds are optimized using a linear approximation of their bunker consumption function.
8. The hotel cost is computed for each vessel from its phase-out time until its phase-in time, except during SOS opportunities.

The goal of this problem is solely to minimize cost; no revenue earning components are taken into consideration in a way that results in the overall cost of an activity from becoming negative. In other words, we allow equipment to be carried, which is a revenue earning activity. However, we model the carrying of equipment through the reduction of sailing costs, and ensure that those costs can never be negative. Despite its simplifications, the NCLSFRP has a number of interesting and difficult to solve

components. In particular, the interactions between the vessels and the time-dependent task costs differentiate the NCLSFRP from other problems.

The NCLSFRP lies on the border of planning and scheduling, and it is not clear a priori which type of method should be used to solve it. Scheduling is concerned with the optimal allocation of scarce resources to activities over time [76]. Compared with AI planning in general, scheduling research has focused on problems that only involve a small, fixed set of choices, while planning problems often involve cascading sets of choices that interact in complex ways [135]. Another limitation is that mainstream scheduling research has focused predominately on the optimization of selected, simple objective criteria such as minimizing makespan or minimizing tardiness [136].

To this end, we solve the NCLSFRP with a variety of methods, including mixed-integer programming (MIP), constraint programming (CP), and automated planning. In our quest to find good solutions to the NCLSFRP, we even go so far as to extend automated planning with a optimizing, branch-and-bound version of partial-order planning (POP) called Temporal Optimization Planning (TOP). Although ultimately our constraint programming approach has the best performance, our results for TOP are very encouraging and suggest that planning is capable of solving difficult combinatorial optimization problems when they have specific properties.

In this chapter, we first describe the dataset we use to test our approaches in Section 5.1, which consists of 11 instances modeling a real-world repositioning scenario from our industrial collaborator. We provide four models of the NCLSFRP and compare them on our dataset. We model the NCLSFRP in PDDL, a domain-specific language for automated planning in Section 5.2, in the novel planning paradigm TOP in Section 5.3, with mixed-integer programming in Section 5.4, and with constraint programming in Section 5.5.

5.1 Dataset

We created eleven instances based on the Asia-CA3 case study shown in Section 3.4. We generated a single instance based closely on a slightly modified version of the actual repositioning scenario faced by our industrial collaborator. In this instance, there are three Panamax sized vessels on three different services in Maersk Line's network. The instance takes place over an 11 week period with the goal of phasing in all three vessels to the Intra-WCSA service. There are three SOS opportunities available using the Asia-CA3 service, in weeks 3, 4 and 5 of the model. These correspond to times when the repositioning coordinator can afford to lose the backhaul of the Asia-CA3 due to the repositioning, if the SOS is chosen.

The SOS was modeled using three start ports: Yantian, China; Yokohama, Japan; or Hong Kong, China. The end port was designated as Balboa, Panama, as this port is a well-known "turn-port" on the Asia-CA3 service. That is, the vessel tends to offload all or almost all of its cargo at Balboa before continuing back towards Japan and China. Sail equipment opportunities are provided from Dalian, China, where refrigerated con-

tainers are in surplus, to several ports in South America: Buenaventura, Colombia; Callao, Peru; and Arica; Chile.

Given this instance, we then varied the opportunities available, as well as the number of vessels, in order to gauge the scaling performance of our approaches. The instances contain between one and three vessels and various combinations of SOS, equipment opportunities and cabotage restrictions. We adopt a naming scheme to identify instances where the instance AC3_v_sce is an instance with v vessels and s SOS opportunities. The "c" indicates that the instance has cabotage restrictions on some ports in the SOS opportunities, and the "e" means that there are sail equipment actions present.

5.2 A PDDL Model of Fleet Repositioning

We model the NCLSFRP using PDDL in order to utilize existing planners for solving the problem. PDDL [43, 47, 100] is a well-known domain specific language for modeling automated planning problems. It is a domain independent modeling language used by the automated planning community, and has been a mainstay of the International Planning Competition (IPC) since its inception in 1998 [27]. There are a number of different PDDL specifications, many with features not necessary for this dissertation. We use PDDL 2.2 [43] because it is sufficient for modeling the NCLSFRP, and is supported by the planner POPF [33]. A distinctive advantage of PDDL over other problem modeling techniques, such as mixed-integer programming, is the extensibility of models, in which extra actions can often be simply appended to a model. Other modeling paradigms generally require new sets of variables that must then be integrated with existing model components. Additionally, PDDL actions (generally) correspond very closely with real-world activities. A sailing action in PDDL directly maps to the sailing of a real vessel, meaning the models are significantly less abstract and easier for domain experts without expertise in automated planning to understand. This section is based off of our PDDL model description of the NCLSFRP in [146].

5.2.1 PDDL Model

The PDDL model is based around keeping track of the state of the vessel throughout its repositioning, starting from the initial state, when all vessels are sailing on their initial services. During the initial state, we do not need to take any costs into account, as the vessel has not yet started the repositioning. The planner then makes a decision to phase out each vessel from its initial service, putting the vessel into a state of transit. While the vessel is in this state, its activities count towards the overall cost of the repositioning. The planner finally makes a decision as to where the vessels should be phased in and when each vessel should be phased in. Once phased-in to the goal service, the vessels no longer generate any repositioning costs and the goal state of the model is reached.

The PDDL model of the LSFRP [146] has interesting temporal features: required

concurrency [36], timed initial-literals (TILs) [43] and duration-dependent effects. We describe the model at a high level and provide more details, as well as the PDDL model in full, in Appendix A.

Initial and Goal States

The initial state of the PDDL model consists of all vessels being in a non-phased out state, with all repositioning opportunities available for use. The initial state of the NCLSFRP makes use of a PDDL construct called a *timed initial literal* (TIL). TILs allow facts to be added and removed from the plan at specified time points. Thus, instead of only having an initial state specified at time zero, TILs can be thought of as allowing pieces of the initial state to be distributed over the planning horizon. We use TILs to specify when particular actions may be used by a vessel. We do this by using a TIL that introduces a fact that "opens" the action for use at a particular time, and another TIL that deletes the fact in order to close the action when it may no longer be used. For example, allowing a vessel to phase-out at some port p at time 40 would be accomplished through a TIL adding an "allow phase-out" fact at time 40 and deleting it at time 40.1. The goal state of the model requires that all vessels have reached their destination, and that the hotel cost has been calculated.

Actions

The PDDL model of the NCLSFRP requires 6 types of actions: `phase-out`, `phase-in`, `calculate-hotel-cost`, `sail`, `sail-on-service` and `sail-with-equipment`. We create phase-out actions at every port call along each vessel's initial service. Phase-out actions may only be applied for a particular vessel when it is in its initial state, meaning it has not yet begun its repositioning. Applying a phase-out action transitions a vessel to a state of transit. Phase-in actions are created at each port on the goal service in each week of the planning period. Vessels must be in a transit state to use a phase-in action. The effect of a phase-in action indicates that a vessel has reached the goal service, which satisfies the goal state of the model.

While a vessel is in transit (i.e., repositioning), it may use any sailing, SOS or sail equipment action. Sailing actions are created between all ports in the model, except for sailings with phase-out ports as the destination, as once a vessel leaves the phase-out service it is not allowed to go back[1]. We create SOS actions for each SOS specified by the repositioning coordinator, and equipment sailing opportunities between all ports with equipment surpluses and those with equipment demands.

Our actions describing sailing, SOS opportunities, equipment sailings, and hotel cost are *durative* actions, meaning they take place over a time period that is specified through the preconditions of the action. In the case of SOS actions, the amount of time they require is fixed based on the start port and end port of the action. However, sailing with and without equipment has a variable duration that must be between

[1]This is a simplification of the overall LSFRP that we handle in Chapter 6.

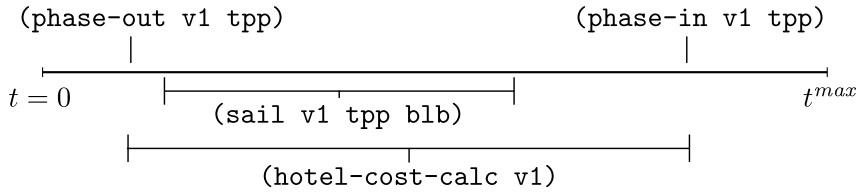


Figure 5.1: An example plan generated by our PDDL model with a hotel envelope action.

the minimum and maximum speed of a vessel. Phase-out and phase-in actions are instantaneous actions that have no duration.

We model sailing costs by setting the fixed cost of sailing (and sailing with equipment) actions to the maximum possible cost of sailing between two ports, i.e., sailing at maximum speed. We then subtract an amount from the action cost based on the duration of the sailing, meaning longer sailing subtract more from the cost, making them cheaper.

All repositioning plans contain, at the very least, a phase-out and a phase-in action for each vessel. Sailings, SOS opportunities, and sail equipment actions can be ordered between the phase-out and phase-in in order to bring the vessel to the goal service.

Hotel period The hotel cost represents a particularly challenging to model aspect of the NCLSFRP. We use a so-called *envelope action*. Such actions are durative and enclose other actions over their duration. In the case of the NCLSFRP, the hotel period for a vessel extends from its phase-out until its phase-in, with the exception of any time spent on an SOS. We compute the hotel cost based on the length of the hotel envelope action.

Figure 5.1 shows an example PDDL plan in which vessel `v1` phases out at port `tpp` and sails to port `blb`, where it phases-in. Notice how the hotel action, `hotel-cost-calc`, encompasses the entire repositioning of `v1`, including the sail action. Of further note is the way the sail action does not require the entire time period between the phase-out and the phase-in. If a vessel is sailing at its minimum speed, as in the case of `v1`, the action cannot take any longer. The vessel must then wait at its destination until performing another action. This waiting time period incurs hotel costs, which are captured with envelope actions.

During SOS opportunities, the hotel cost is not applicable. However, if an SOS opportunity is chosen during the hotel envelope, hotel costs would be erroneously calculated over its duration. We split the hotel period into two envelope actions, one that comes before an SOS, and one that comes after, in order to correctly exclude hotel costs during an SOS. We note that there are other options for modeling such actions, and describe our reasoning in detail in Appendix A.1.5.

Reverse Model

A common practice when writing PDDL models is to create a *reverse model*, in which the effects and preconditions of actions are swapped, as are the initial and goal states. In a reverse model, for example, vessels sail "backwards" from their destination to their source. Although such models become less understandable, they sometimes make models easier for planners to solve. Most modern planners use a forward search paradigm, meaning that models where the difficult decisions come at the end can be hard to solve in a forward model. The LSFRP is one of these problems, as the phase-in activities create a number of interactions between vessels that the planner must resolve. Since the reverse model is equivalent to the forward model, we do not describe it, but refer to Appendix A.3, where the full domain model is provided.

5.2.2 Planners

The time-dependent task costs in the NCLSFRP pose a particular challenge to planners, especially in regards to finding an optimal solution. Envelope actions and numerous TILs pose additional challenges, requiring planners to not only support a broad swath of PDDL 2.1, but also to have considered approaches for dealing with issues that, until now, have not been present in many planning problems. We highlight a number of temporal planners that have cost and makespan optimization components, and identify why they cannot model the NCLSFRP.

IxTeT [53] and ZENO [118] are two notable, early partial-order temporal planners. IxTeT uses a hierarchical planning approach with multi-valued domain attributes and a discretized formulation of time. IxTeT focuses on achieving temporal feasibility of domains, and does not perform any optimization. ZENO, unlike IxTeT, does not rely on a discretized view of time and supports piecewise continuous linear change. Remarkably, ZENO uses a linear programming solver internally for determining the feasibility of time constraints in partial plans, similar to a number of modern planners. However, ZENO ignores the optimization capabilities of linear programming and performs no optimization of problem objectives.

The Sapa planner [40] can perform multi-objective optimization, for example optimizing both makespan and plan cost simultaneously, and provides heuristics for solving such problems. Sapa is not able to prove optimality, nor can it handle time-dependent task costs with a non-fixed duration. The LPG planner [51, 50] plans using stochastic local search on top of a special type of temporal planning graph. Thus, it does not guarantee optimality, and it, like Sapa, is unable to model time-dependent task costs, a critical features of the NCLSFRP.

The CPT planner [161] performs temporal planning using a constraint programming model to provide search-space pruning to partial-order causal link planning. Even though CPT utilizes a constraint programming model to solve problems, the optimization it performs is makespan optimization, rather than optimization of a general objective function. While it is likely that CPT could be modified to work with time-dependent

task costs, such a modification is not trivial due to its impact on the temporal planning heuristics present in the planner.

Recent net-benefit planners (e.g., HSP*, MIPS-XXL and Gamer) [63, 86], while also strong at optimization, suffer from the same problem as Sapa and LPG; they do not allow the duration of an action to influence its cost.

A number of powerful solvers have recently been developed for planning languages with durative actions, real-valued state variables, and linear change of quantities during action execution [31, 95, 132]. These planning languages can model domains where activities depend on shared resources like electric power during execution. This situation is typical for popular application domains within robotics and aerospace systems (e.g., [49, 102]). Most of these domain-independent and application specific planning systems, however, only allow simple objective criteria like makespan minimization.

Recently, a number of planners have taken an increased focus on scheduling and problems with continuous numeric change. The planners Colin [32], POPF [33], Kongming [95] and TMLPSAT [132] all use an LP or MIP internally to model continuous change. This LP or MIP can also be used to model continuous change that is related to action durations. Colin and its successor POPF are the only two available planners that can reason with all the necessary language features for the NCLSFRP. Kongming does not allow multiple parallel updates to the same variable, but here total cost is updated by multiple vessels sailing in parallel, and TMLPSAT does not exist in a runnable form. Colin and POPF are not optimal planners, though POPF can continue searching once a solution is found to improve its quality [30]. The OPTIC planner from [11] extends the POPF planner to support preferences and time-dependent task costs, but offers no new capabilities for the NCLSFRP not already present in POPF.

5.2.3 POPF

We use the planner POPF to solve the NCLSFRP PDDL domain, as it is the only planner currently capable of doing so. We briefly describe how POPF solves problems and the features relevant to the NCLSFRP, and then describe several extensions that we have made to POPF (and published in [147]) in order to better orient the planner towards problems like the NCLSFRP.

POPF is based on the COLIN planner [31, 34], which is itself based on the CRIKEY3 planner [29]. All of these planners split durative actions into two instantaneous ("snap") actions representing the start and the end of the durative action. The starting action contains the preconditions and effects at the start of the durative action, and the end action has the preconditions and effects at the end of the action. The goal state is modified such that a plan is not considered complete unless all start actions in the plan have a corresponding end action. Preconditions that must hold over the entire action require special handling, but since we do not use this for the NCLSFRP, we refer readers to [34] for an overview.

At a high level, the search procedure of POPF works as follows. Actions are assigned a plan step in which they occur, which itself is assigned a timestamp from a scheduling

algorithm in the case of temporal planning. Actions are added to the plan to build a plan from the initial state. Given a particular state not satisfying the conditions of the goal state, POPF must decide which action to apply to the current state based on the available actions. The planner makes these decisions using an enforced hill climb [66], which is a hill climbing (i.e., greedy local search) algorithm that enlists the assistance of a breadth first search to choose good neighbors. Once a solution is found, POPF switches to a weighted A* (WA*) with $w = 5$ to try to find improvements. POPF can be set to find an optimal plan by using the A* algorithm from the start, however this is not considered a standard use case of the planner.

Search guidance is provided by the *temporal relaxed planning graph* (TRPG), which is an extension from [29] of the well-known planning graph heuristic from the GRAPHPLAN planner [14]. The graph can be used to estimate which actions are necessary to complete a partial plan. After decision to add an action to add to the plan, POPF checks the plan for temporal feasibility and updates its continuous change variables. The planner continues this process until it has found a complete plan. We now describe the components of POPF's search in more detail, along with the extensions necessary to make POPF more effective at solving the NCLSFRP.

Temporal Consistency and Continuous Change

Temporal feasibility and continuous change are implemented in POPF through an LP that is built off of a given state (a partial plan). Minimum and maximum action durations are enforced in POPF using LP constraints that ensure the plan step with the beginning of a durative action and the end of the same action are at least (or at most) the necessary time units apart from each other. Each plan step is required to have a time step of at least the step before it, plus a small epsilon value. Note that in contrast to COLIN, which imposes a total order over the actions in the plans it finds, POPF has no such restriction. Rather, POPF only imposes a partial-order over the actions as in partial-order planning, but still performs a forward planning procedure. POPF is thereby able to combine the temporal advantages of partial-order planning with the powerful heuristics that have been developed for forward planning. In addition to the temporal aspects of the plan, POPF models continuous (linear) change. It does this in the LP using decision variables representing the value of a continuous planning variable at each step where the variable is modified or needed. Next, we define the LP used by POPF formally.

Parameters We are given a plan with n steps and m durative actions. Let the function $start : \{1, \ldots, m\} \to \{1, \ldots, n\}$ ($end : \{1, \ldots, m\} \to \{1, \ldots, n\}$) return the step of the start (end) action corresponding to a particular durative action. Each durative action $j \leq m$ has a minimum duration Δ_j^{min} and maximum duration Δ_j^{max}. Let v be the number of continuous planning variables in the plan (i.e., those variables that are modified by continuous change in the PDDL model). To avoid confusion, we refer to variables in the LP as decision variables, whereas the variables in the PDDL

model of a planning problem are referred to as continuous planning variables. We use the function $prev : \{1, \ldots, v\} \times \{1, \ldots, n\} \to \{1, \ldots, n\}$ to provide the previous step in which a planning variable was modified.

The rate of change for a planning variable over time is given by δ_i, where $1 \leq i \leq v$. The action (or actions) in a given step can modify the planning variables in that step. In each step of the plan, planning variables can be the subject of action effects. Let $\theta(i, j)$ be the set of planning variables that are used to update variable i in step j, i.e., $\theta : \{1, \ldots, v\} \times \{1, \ldots, n\} \to 2^v$. Let β_{ijl} be the coefficient associated with each planning variable $l \in \theta(i, j)$, with i as the planning variable being modified and j the plan step. Actions can require a linear combination of planning variables to respect a given lower and upper bound. Let κ_j be the number of constraints in step j and $\phi(l)$ be the set of planning variables in action precondition $l \leq \kappa_j$. We assign a coefficient α_{ijl} to each planning variable i in step j with constraint $l \leq \kappa_j$. Each constraint over the planning variables at each step is associated with a lower, ℓ_{jl}, and upper bound, u_{jl}. We summarize the parameters in the following table.

v	Number of continuous planning variables present, indexed by i.
n	Number of steps in the plan, indexed by j.
m	Number of durative actions in the plan, indexed by k.
$start(k)$	Gives the start step of durative action $k \leq m$.
$end(k)$	Provides the end step of durative action $k \leq m$.
$\Delta_k^{min}, \Delta_k^{max}$	Minimum and maximum duration of durative action $k \leq m$, resp.
$prev(i)$	Supplies the previous step in which a planning variable $i \leq v$ was modified by an action.
δ_i	Rate of change of planning variable $i \leq v$.
$\theta(i, j)$	Set of planning variables used in the update of variable $i \leq v$ in step $j \leq n$.
β_{ijl}	Coefficient of planning variable $l \in \theta(i, j)$.
κ_j	Number of constraints over planning variables at step $j \leq n$.
$\phi(l)$	Set of planning variables used in action precondition $l \leq \kappa_j$.
α_{ijl}	Coefficient of variable $i \leq v$ in step $j \leq n$ constraint $l \leq \kappa_j$.
ℓ_{jl}, u_{jl}	Lower and upper bound of step $j \leq n$ constraint $l \leq \kappa_j$, resp.

Decision variables There are three types of decision variables in the model. First, the decision variables $t_i \in \mathbb{R}^{0+}$ represents the time of plan step $i \leq n$. Next, we let the decision variables x_{ij}, x'_{ij} be the value of planning variable i directly before plan step j and directly after, respectively.

Constraints Using the parameters and decision variables above, we define an LP with no objective representing the temporal components of POPF along with its continuous planning variables.

$$t_j + \epsilon \leq t_{j+1} \qquad \forall\, 1 \leq j < n \tag{5.1}$$

$$\Delta_k^{min} \leq t_{end(k)} - t_{start(k)} \leq \Delta_k^{max} \qquad \forall\, 1 \leq k \leq m \tag{5.2}$$

$$x_{ij} = x'_{i,prev(i)} + \delta_i(t_i - t_{prev(i)}) \qquad \forall\, 1 \leq i \leq v, 1 \leq j \leq n \tag{5.3}$$

$$x'_{ij} = x_{ij} + \sum_{l \in \theta(i,j)} \beta_{ijl} x_{lj} \qquad \forall\, 1 \leq i \leq v, 1 \leq j \leq n \tag{5.4}$$

$$\ell_{jl} \leq \sum_{i \in \phi(l)} \alpha_{ijl} x_{ij} \leq u_{jl} \qquad \forall\, 1 \leq j \leq n, 1 \leq l \leq \kappa_j \tag{5.5}$$

$$t_j \geq 0 \qquad \forall\, 1 \leq j \leq n \tag{5.6}$$

Constraints (5.1) ensure each step has a time at least that of the step before it, and constraints (5.2) force the steps of the plan to obey the minimum and maximum duration of durative actions. Continuous planning variables have their values before the step updated in constraints (5.3), and in constraints (5.4) the values after the step are updated with the effects of the step. In constraints (5.5), linear combinations of the planning variables are required to be within specified upper and lower bounds in order for a step of the plan to begin. Finally, constraints (5.6) post the bounds of the time step decision variables. In the event that this LP is infeasible, it means the current plan is also infeasible and can be discarded.

The objective function of the above LP can take several forms. In the case of makespan minimization, the objective is simply to minimize t_n, the time of the last action in the plan. For a cost based objective, the objective can be to minimize $\sum_{i=1}^{v} \sum_{j=1}^{n} \gamma_{ij} x'_{ij}$, where γ_{ij} is the coefficient of variable i in step j. This objective allows planning variables' values to be taken into account after each planning step.

The Temporal Relaxed Planning Graph

The TRPG is used to estimate which actions will be necessary to complete a plan, thereby providing a lower bound on the amount of time or cost that the plan will require. The TRPG heuristic used in POPF was first introduced in the CRIKEY3 planner as an extension to the TRPG present in Sapa [40] and CRIKEY [28]. The TRPG is based off of a *relaxed planning graph* (RPG) with temporal components. The RPG heuristic was first introduced in [67] in the FF planner. The RPG heuristic works by building a so-called *planning graph* using a *delete relaxation* of the original problem. The delete relaxation involves omitting the delete list from each action, i.e., no facts can be removed from the plan.

The planning graph consists of alternating layers of actions and facts, starting with a fact layer representing the initial state. That is, a *fact node* is created for each fact in the initial state (ignoring TILs). Without going into detail about the algorithm for generating a planning graph, which is not necessary for this dissertation, we describe the steps for creating the relaxed planning graph. After the initial fact layer, an action layer

is applied in which nodes are created for each action which has all of its preconditions supported by the facts available in the initial state. In addition, a *no-op* node is created in the action layer for each fact in the initial layer. The no-op node represents an action with a precondition and effect for a particular fact. The no-op nodes allow facts to persist through the layers of the planning graph. Next, a new fact layer is appended containing a fact node for each fact present in the actions in the previous action layer, including no-op nodes. The algorithm iterates creating a fact layer and then an action layer until a fact layer contains all of the facts present in the goal state. At this point, a relaxed plan can be extracted from the graph by searching in reverse from the goal state in which a set of actions are pulled from each layer of the graph.

We now explain how the RPG turns into the TRPG. First, each RPG layer is assigned a timestamp determined through using the LP to provide a lower bound for achieving the numeric preconditions for that layer. Second, TILs are included as dummy actions that can only be applied at a specific time. When a TIL deletes a fact, the fact is prevented from being used at a later time in the plan, unless it is independently re-added. Furthermore, temporal separation between layers is introduced based on the continuous change present. Finally, POPF makes several inferences about the time particular steps can occur based on the deletion and re-adding of facts, and prevents actions from using these facts at times they would not be present in the plan.

TIL Multiple Time Window Abstraction

In POPF, TILs have traditionally been treated as dummy actions that must be applied at the time when the TIL fact becomes available. That is, at each point in search, the planner can choose to apply an action or the next TIL. This is a sensible approach to TIL handling when the number of TILs is not that large, but in the NCLSFRP the number of TILs can be over 100. A dummy action representation of TILs results in a significant increase in the size of the search space for the NCLSFRP, since at any decision point there are many subsequent TILs that can be applied.

To this end, we introduce in [147] a domain-independent technique for abstracting TILs in two circumstances. Both situations involve a TIL adding a fact that begins a time window in which the fact is available, and a TIL deleting the same fact, thus ending the time window. The first situation involves a single fact that has multiple time windows in which it is available. The second also involves multiple time windows, but all actions that use the fact delete it immediately after its use. For example, in the phase-in for the NCLSFRP, a phase-in time slot is opened at each port each week, but once it is used, the time slot is no longer available.

We model the above TIL situations by converting POPF's LP into a MIP through the use of binary variables as follows. Given the set of facts with multiple TIL time windows, F, and a fact $f \in F$, let ω_f be the number of time windows for f. Each time window has a beginning time defined by $\mathit{begin}(f, w)$ and an end time $\mathit{end}(f, w)$, where $f \in F$ and $1 \leq w \leq \omega_f$. We introduce the binary decision variable $y_{iwf} \in \{0, 1\}$ which is set to 1 if step $i \leq n$ occurs within time window $w \leq \omega_f$ of fact $f \in F$. To handle

the second case of TILs which are removed by planning actions, we let $F' \subseteq F$ be the set of facts that are deleted by all actions that require that fact in their precondition. The set $S_f \subseteq \{1, \ldots, n\}$ contains all of the steps that refer to fact $f \in F'$.

$$\sum_{w=1}^{\omega_f} begin(f, w) y_{iwf} \leq t_i \leq \sum_{w=1}^{\omega_f} end(f, w) y_{iwf} \qquad \forall\, f \in F, 1 \leq i \leq n \tag{5.7}$$

$$\sum_{w=1}^{\omega_f} y_{iwf} = 1 \qquad \forall\, f \in F, 1 \leq i \leq n \tag{5.8}$$

$$\sum_{i \in S_f} y_{iwf} \leq 1 \qquad \forall\, f \in F', 1 \leq w \leq \omega_f \tag{5.9}$$

Constraints (5.7) limit the time of a particular step to be within a particular fact time window, if the time window is chosen through the variables y_{iwf}. Constraints (5.8) ensure that only a single time window per step is chosen for a particular fact. We require that if a fact is deleted immediately after being used, that it be used in at most one step of the plan in constraints (5.9).

The TRPG heuristic must also be modified to take into account TILs that are not handled as dummy actions. The modification works as follows. Consider an action with a set of preconditions, some of which are provided by TILs, and some which are not. Given a layer t of the TRPG where the non-TIL provided preconditions are true, the action is added in a layer $t' \geq t$ such that the TIL provided preconditions are true. In this way, the TRPG is actually no longer ignoring the deletion of certain TILs. In the standard TRPG, a TIL is added at a particular time and the fact is then "stuck" in the subsequent layers of the graph. However, by avoiding the conversion of TILs into dummy actions, the TRPG need not have the facts of TILs represented explicitly by nodes.

Cost Estimation

Time-dependent task costs pose a particular challenge for POPF, since, although it supports them, it was not designed with them in mind. Thus, we make the following modification to the heuristic when time-dependent task costs are present. We use a bounding procedure which prevents any actions from being appended to a plan if it would increase the plan's cost to be greater than the current incumbent solution cost. Formally, given a solution with cost c, and a state s with cost c' (which must be less than c), any action with a minimum cost greater than $c - c'$ will result in a solution worse than the current incumbent. Note that this assumes that the total plan cost is monotonically increasing as actions are added to it, an assumption that holds for the NCLSFRP and can be easily checked.

5.2.4 PDDL Model Computational Evaluation in POPF

We ran the POPF planner on our NCLSFRP dataset (see Section 5.1 for an overview) with and without the improvements discussed above. We used AMD Opteron 2425 HE processors with a maximum of 4GB of RAM with CPLEX 12.3. We use this experimental setup throughout the chapter. We perform both optimal planning, in which POPF uses an A* algorithm to find optimal solutions, as well as satisficing planning, in which POPF attempts to improve the objective value of the solutions it finds heuristically. Table 5.2 shows the CPU times and solution gaps in parenthesis (when satisficing). We compute the optimality gap with the formula $(c - c^*)/c^*$, where c^* is the optimal plan cost and c is the cost of the best solution found. Thus, a gap of 0 means the solution found is optimal, and the larger the optimality gap, the worse the solution's cost.

We perform optimal planning with both the forward and reversed domain, without much success. POPF only solves two instances using forward search, and three using backwards search. Additionally, these are some of the smallest instances in our dataset. Given that POPF was not designed for finding optimal plans these results are not particularly surprising, and show that more work is needed in the area of optimal planning for problems like the NCLSFRP.

We carry out satisficing planning using several different configurations of POPF, all of which use the improved cost estimation technique. The "standard" setup uses TIL time window abstractions and solves the resulting MIP problem to optimality only at goal nodes. The LP relaxation is solved at all other search nodes, which saves significant amounts of CPU time at non-goal nodes. We can do this because the LP is sufficient to determine temporal infeasibility and provides a reasonable lower bound on the cost of the plan. Additionally, all continuous planning variable effects are modeled through the LP, meaning they are unaffected by relaxing the integer variables. We then perform a makespan optimization to show that the NCLSFRP cannot be solved effectively using such techniques ("Makespan" column), and we also use the "standard" configuration without using the MIP relaxation ("No-MIP-Relax"). That is, we solve the MIP to optimality at each node. Next, we also use POPF without the TIL abstraction techniques introduced to evaluate their effectiveness ("No-TIL-Abs"). Finally, we use the "standard" configuration of POPF on the reversed domain ("Reversed").

Standard POPF finds the lowest gap, or ties with the reversed domain, on all but one of the instances. The MIP relaxation technique used in "Standard" provides better solutions than all other POPF configurations except on one instance (AC_3_2_2ce), where it is outperformed by turning off MIP relaxation. Disabling TIL abstractions ("No-TIL-Abs") results in POPF losing the ability to find good solutions, with the optimality gaps often twice as high as for the standard configuration. Finally, using makespan as an optimization criterion results in poor plans with high costs. This is not surprising since minimizing makespan means vessels will be sailing at their maximum speeds, which is expensive.

Although POPF succeeds in finding optimal solutions on smaller instances (AC3_1_0 – AC3_1_1e), and coming within two to three times the optimal solution cost on the

Inst.	POPF (Optimal)		POPF (Satisficing)									
	Forwards	Reversed	Standard		Makespan		No MIP relax		No-TIL-Abs		Reversed	
AC3_1_0	0.7	1.4	0.4	(0.0)	0.1	(1.7)	0.7	(0.0)	105.8	(0.0)	0.4	(0.0)
AC3_2_0	-	809.6	32.5	(0.0)	3.2	(1.6)	113.2	(0.0)	13.0	(0.1)	78.1	(0.0)
AC3_3_0	-	-	1105.1	(0.0)	117.5	(2.3)	3041.6	(0.0)	88.2	(0.1)	39.2	(0.8)
AC3_1_1e	3.3	4.0	1.7	(0.0)	0.1	(0.7)	2.3	(0.0)	1079.3	(0.3)	1.2	(1.2)
AC3_2_2ce	-	-	1550.6	(0.3)	1.1	(19)	2284.2	(0.0)	31.3	(3.7)	892.5	(1.6)
AC3_3_2c	-	-	399.2	(0.2)	9.2	(7.3)	26.3	(1.4)	303.4	(1.3)	602.8	(1.1)
AC3_3_2e	-	-	291.5	(1.3)	9.6	(11)	28.4	(2.4)	310.8	(2.3)	688.6	(1.9)
AC3_3_2ce1	-	-	303.9	(1.3)	9.7	(11)	28.4	(2.4)	314.5	(2.3)	697.2	(1.9)
AC3_3_2ce2	-	-	1464.2	(1.6)	10.0	(12)	204.9	(2.8)	303.4	(2.7)	690.1	(2.3)
AC3_3_2ce3	-	-	348.0	(1.1)	10.3	(8.7)	29.4	(1.9)	308.4	(1.7)	603.3	(1.5)
AC3_3_3	-	-	1975.5	(2.3)	10.1	(15)	226.0	(3.6)	352.6	(3.4)	699.6	(2.9)

Table 5.2: CPU times in seconds with optimality gaps in parenthesis for POPF on the NCLSFRP dataset.

rest of the instances, there is still room for improvement. Furthermore, a certificate of optimality is desirable in order to assure repositioning coordinators of the quality of the answer they are being provided by the planning system. POPF is unable to provide such a certificate unless it is set to optimal planning, which results in less instances solved to optimality. One of the main difficulties for POPF (especially in the case of optimal planning) is that it spends too much time examining permutations of `hotel-cost-calc` actions. Additionally, different orderings of `sail` actions are also considered with `hotel-cost-calc` actions, perhaps indicating difficulties in dealing with both envelope actions and time-dependent task costs in a single planning model. New solutions and methods are required in order to solve the NCLSFRP to optimality across the entire dataset.

5.3 Temporal Optimization Planning

Planning techniques offer strong heuristics and solving capabilities for the selection and sequencing of activities with logical preconditions and effects in order to achieve some set of goals. Such approaches seem like a natural fit for combinatorial optimization problems with activity selection components, especially those like the NCLSFRP, with time dependent task costs, where non-activity based formulations in, e.g., mixed-integer or constraint programming, become mired in logical (i.e., if-then) constraints.

However, despite the many planners that exist for classical and temporal planning domains, very few planners are able to support problems like the NCLSFRP, due to a focus on satisfaction and fixed cost objective optimization within the planning community. In fact, the POPF planner [33] is the only planner capable of handling the repositioning domain we propose, but as is clear from the computational evaluation of POPF in Section 5.2.4, solving NCLSFRP problems to optimality is difficult for POPF. Given the large amount of money that can be saved through effective repositioning, finding new methods of solving combinatorial optimization problems with time-dependent task costs is a worthwhile endeavor.

To this end, we introduce Temporal Optimization Planning (TOP), which was first described in [147] and [148]. The main contribution of TOP is to use automated planning to build and search through optimization models that involve continuous time, metric quantities, and a complex mixture of action choices and ordering constraints. The need to solve such problems is not new. TOP fundamentally diverges from classical automated planning approaches by introducing two sets of modeling variables that decouple the planning problem from the optimization model. Thus, the optimization model is not tightly bound to the semantics of actions. Actions are merely used as handles to optimization components that are combined to complete optimization models using partial-order planning. In other words, TOP builds optimization models with the help of automated planning, i.e., the focus of TOP is on the optimization model of a problem, whereas classical planning approaches include optimization components within actions merely as a byproduct of the overall planning process.

TOP is built on a state variable representation of propositional STRIPS planning [46]. TOP utilizes partial-order planning [117], and extends it in several ways. First, an optimization model is associated with each action in the planning domain. This allows for complex objectives and cost interactions that are common in real world optimization problems to be easily modeled. TOP can utilize any optimizer, including linear programming, mixed-integer programming and constraint programming solvers. Second, instead of focusing on simply achieving feasibility, TOP minimizes a cost function. Finally, begin and end times can be associated with actions, making them durative. Such actions can have variable durations that are coupled with a cost function.

TOP differs from existing temporal planners in two further ways. First, TILs are not needed to model problems in which some actions are only available at specific times, such as the `phase-out` and `phase-in` actions in the NCLSFRP. Rather, constraints on the start or end time of an action can be built directly into actions' optimization models and exploited for search guidance. Second, through shared variables in their optimization models, actions can refer directly to start/end times of other actions. This means the encoding of, e.g., the hotel cost calculation can be embedded within the effects of other actions that imply it. PDDL actions cannot directly refer to start/end times of other actions, which is why the PDDL model must use envelope actions to model the hotel cost. Such actions result in an expansion of the search space.

We extend the formulation of partial-order planning presented in Section 4.2 to include the optimization components that set TOP apart from other planning paradigms. Formally, a TOP problem is represented by a tuple

$$P = \langle \mathcal{V}, \mathcal{D}, \mathcal{A}, \mathcal{I}, \mathcal{G}, \mathit{pre}, \mathit{eff}, \boldsymbol{x}, \mathit{obj}, \mathit{con} \rangle,$$

where $\mathcal{V} = \{v_1, \ldots, v_n\}$ is the set of state variables, $\mathcal{D}$ is the Cartesian product of the domains $D(v_1) \times \cdots \times D(v_n)$, $\mathcal{A}$ is the set of actions, $\mathcal{I}$ is a total state variable assignment (i.e., $\mathit{vars}(\mathcal{I}) = \mathcal{V}$) representing the initial state, $\mathcal{G}$ is a partial assignment (i.e., $\mathit{vars}(\mathcal{G}) \subseteq \mathcal{V}$) representing the goal states, pre_a is a partial assignment representing the precondition of action a, eff_a is a partial assignment representing the effect of

action a, $\boldsymbol{x} \in \mathbb{R}^m$ is a vector of optimization variables[2] that includes the begin and end time of each action, x_b^a and x_e^a respectively, for all actions $a \in A$, $obj_a : \mathbb{R}^m \to \mathbb{R}$ is a cost term introduced by action a, and $con_a : \mathbb{R}^m \to \mathbb{B}$ is a constraint expression introduced by action a with $con_a \models x_b^a \leq x_e^a \wedge x_b^a \geq 0 \wedge x_e^a \geq 0$. Our formulation of TOP is general enough to encompass any type of objective function, be it linear or non-linear, as well as nearly any type of constraint.

We define $M_a = \min\{obj_a \mid con_a\}$, which is the minimal cost of action a's optimization model component. In other words, M_a is the result of minimizing obj_a subject to the constraints con_a.

We extend the notion of a partial-order plan to include an optimization model that is built as the plan receives new actions and ordering constraints. A temporal optimization plan is represented by a tuple $\langle A, C, O, M \rangle$, where A is the set of actions in the plan, C is a set of causal links $a \xrightarrow{\mu} b$ with $a, b \in A$ and $\mu \in eff_a \cap pre_b$, O is a set of ordering constraints of the form $a \prec b$ with $a, b \in A$, and M is an optimization model associated with the plan defined by

$$\min \quad \sum_{a \in A} obj_a(\boldsymbol{x}) \tag{5.10}$$

$$\text{s.t.} \quad x_e^{a_i} \leq x_b^{a_j} \qquad \forall a_i \prec a_j \in O \tag{5.11}$$

$$con_a(\boldsymbol{x}) \qquad \forall a \in A. \tag{5.12}$$

The objective of M (5.10) is to minimize the sum of the costs introduced by actions, subject to action orderings (5.11) and the constraints associated with each action in π (5.12). We require this optimization model to be valid for any valid partial plan. Let $cost(\pi)$ be the cost of an optimal solution to M for a partial plan π.

We define open conditions and unsafe links to be the same as in standard partial-order planning. However, to deal with durative actions in TOP we need to keep track of another type of flaw called *interference*. We adopt an interference model based on the exclusive right to state variables [130]. Thus, two actions a and b interfere if $vars(eff_a) \cap vars(eff_b) \neq \emptyset$ and O implies neither $a \prec b$ nor $b \prec a$. Similar to unsafe links, an interference between actions a and b can be fixed by posting either $a \prec b$ or $b \prec a$ to O. The idea behind interferences is that we want to prevent multiple actions from modifying the same state variables at the same time.

Together, open conditions, unsafe links and interferences constitute flaws in a plan. Let $flaws(\pi) = open(\pi) \cup unsafe(\pi) \cup interfere(\pi)$ be the set of flaws in the plan π, where $open(\pi)$ is the set of open conditions, $unsafe(\pi)$ is the set of unsafe links, and $interfere(\pi)$ is the set of interferences. As in the case of standard POP, we say that π is a *complete plan* if $|flaws(\pi)| = 0$, otherwise π is a *partial plan*. A plan π^* is optimal if it is feasible and for all feasible solutions π, $cost(\pi^*) \leq cost(\pi)$.

Figure 5.2 shows an example TOP plan with 3 actions (excluding the initial and goal dummy actions). The plan is a complete plan, as all goals are satisfied and there

[2]We sometimes let $\boldsymbol{x}$ denote a set rather than a vector.

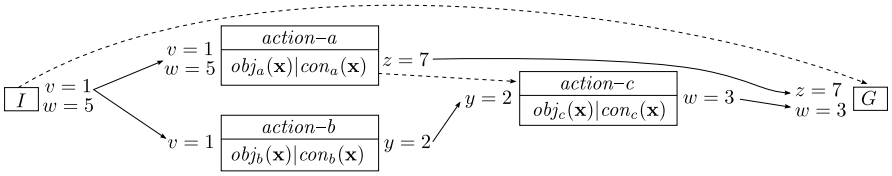


Figure 5.2: An example TOP plan with three actions and the start and end actions I and G, respectively. Each action is shown in a box with preconditions to the left, effects to the right, and the optimization model inside. Solid arcs represent causal links, and dashed arcs ordering constraints.

are no flaws. Notice the ordering constraint (dashed line) between action a and c, due to the interference on the w planning variable. The matching optimization model is:

$$\begin{aligned}
\min \quad & obj_a(\mathbf{x}) + obj_b(\mathbf{x}) + obj_c(\mathbf{x}) \\
\text{s.t.} \quad & con_a(\mathbf{x}) \wedge con_b(\mathbf{x}) \wedge con_c(\mathbf{x}) \\
& x_a^E \leq x_c^B,\ x_b^E \leq x_c^B \\
& x_I^E \leq x_i^B,\ x_i^E \leq x_G^B,\ x_i^B \leq x_i^E \qquad \forall i \in \{a, b, c, I, G\}
\end{aligned}$$

The variables x_i^B and x_i^E represent the begin and end time of action $i \in \{a, b, c, I, G\}$, respectively. The optimization model combines all of the action objectives into a unified objective for the plan. The constraints are also joined together into a conjunction, followed by the ordering constraints that directly follow from the causal links and ordering constraints in the plan.

5.3.1 Linear Temporal Optimization Planning

The TOP formalism provides a theoretical model for solving planning problems with complex objectives and constraints, but in order to use TOP in practice it is necessary to find a way of combining automated planning and optimization such that the problems are actually solvable. In the case of the NCLSFRP, only linear objectives and constraints are necessary for solving the problem.

To this end, we introduce linear temporal optimization planning (LTOP). In LTOP, all of the optimization models associated with planning actions have a linear cost function and a conjunction of linear constraints. Thus, obj_a is of the form $\mathbf{c^a x'}$, where $\mathbf{c^a} \in \mathbb{R}^m$ and con_a is of the form $\bigwedge_{1 \leq i \leq n_a} (\boldsymbol{\alpha}_i^a \boldsymbol{x'} \leq \beta_i)$, where $\boldsymbol{\alpha}_i^a \in \mathbb{R}^m$, $\beta_i \in \mathbb{R}$, and n_a is the number of constraints associated with action a. Thus, M_a and M are LPs. Note that M is very similar to the LPs in POPF, and serves a similar purpose: enforcing temporal constraints and optimizing cost.

Figure 5.3 shows an example LTOP plan for the repositioning in the Asia-CA3 case study from Section 3.4. The optimization variables $h_{b,v}$ and $h_{e,v}$, representing the begin and end of the hotel period, respectively, are of particular note, as they replace the action `calculate-hotel-cost` required by our PDDL model. Each action updates the upper bound of $h_{b,v}$, this shared variable allows the hotel cost of the vessel to be

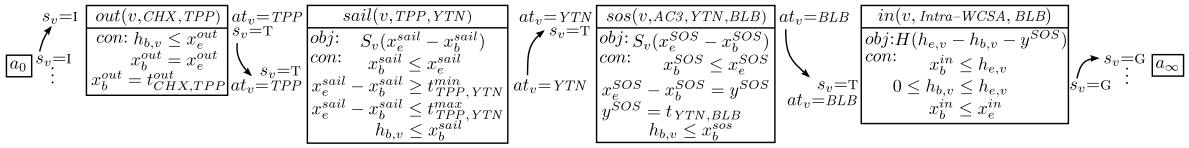


Figure 5.3: A complete TOP plan showing a solution to the repositioning in the Asia-CA3 case study. Boxes represent actions and contain their associated optimization models. Causal links are shown with arrows. The optimization variables x_b^a and x_e^a represent the begin and end time of action a, and $h_{b,v}$, $h_{e,v}$ are the begin and end hotel time of the vessel, respectively. The state variable $s_v \in \{\text{I}, \text{T}, \text{G}\}$ represents the vessel being on its initial service, in transit or at its goal service, respectively.

Algorithm 5.1 Temporal optimization planning algorithm.

```
1:  function TOP(I, G)
2:      Π ← {InitialTOP(I,G)}
3:      π_best ← null
4:      u ← ∞                                  ▷ Cost of the incumbent (upper bound)
5:      while Π ≠ ∅ do
6:          π ← SelectPlan(Π)
7:          Π ← Π \ {π}
8:          if NumFlaws(π) = 0 ∧ Cost(π) < u then
9:              u ← cost(π)
10:             π_best ← π
11:         else if EstimateCost(π) < u then
12:             f ← SelectFlaw(π)
13:             Π ← Π ∪ RepairFlaw(π, f)
14:     return π_best
```

accounted for, even in a partial plan. An example of implicit TIL handling within LTOP can be seen in the *out* action. The starting time of the action is bound to $t_{CHX,TPP}^{out}$, which is a constant representing the time the vessel may phase out at port TPP.

Algorithm 5.1 shows a branch-and-bound algorithm that finds an optimal plan to an LTOP problem, based on the POP algorithm in [164]. First, an initial plan π_{init} is created by the InitialTOP function (line 2). We define $\pi_{init} = \langle\{a_0, a_\infty\}, \emptyset, \{a_0 \prec a_\infty\}, M_{init}\}\rangle$, where a_0 is an action representing $\mathcal{I}$ with $pre_{a_0} = \emptyset$ and $eff_{a_0} = \mathcal{I}$; a_∞ is an action representing $\mathcal{G}$ with $pre_{a_\infty} = \mathcal{G}$ and $eff_{a_\infty} = \emptyset$; and M_{init} is an optimization model with no objective and two constraints, con_{a_0} and con_{a_∞}, which are special constraints on the dummy actions a_0 and a_∞ such that $con_{a_0} = (x_b^{a_0} = x_e^{a_0} \wedge x_b^{a_0} \ge 0)$ and $con_{a_\infty} = (x_b^{a_\infty} = x_e^{a_\infty} \wedge x_b^{a_\infty} \ge 0)$. The optimization variables $x_b^{a_0}, x_e^{a_0}, x_b^{a_\infty}$ and $x_e^{a_\infty}$ represent the begin and end times of actions a_0 and a_∞ respectively. The algorithm then selects a plan from Π (line 6) and checks if it is a complete plan. If so, its cost is compared with the current upper bound (u), and if the cost is lower, the *incumbent* π_{best} is replaced with the current plan π and the upper bound is updated (lines 9 and 10). When π is a partial plan, an estimated lower bound of the plan is computed: if

it is higher than the cost of the incumbent solution, the plan is discarded (line 11). Otherwise, a flaw is selected and repaired (lines 12 and 13). This process is repeated until $\Pi = \emptyset$, then the current incumbent is returned, if one exists.

Algorithm 5.1 is guaranteed to find the optimal solution (if there is one) as long as EstimateCost does not overestimate the cost of completing a partial plan. To prune as much of the branch-and-bound tree as possible, we need tight lower bounds. If we require that the cost of each action subject to its constraints is non-negative, we can prove that $cost(\pi)$ is such a lower bound.

Proposition 5.1. *Given any valid partial plan* $\pi = \langle A, C, O, M \rangle$ *where* $M_a \geq 0$, $\forall a \in A$, $cost(\pi) \leq cost(\bar{\pi})$ *for any completion* $\bar{\pi}$ *of* π.

Proof. Let π' be π with a single flaw repaired. The flaw is either *i*) an unsafe link, *ii*) an interference, *iii*) an open condition being satisfied by an action in the plan, or *iv*) an open condition being satisfied by an action not in the plan.

In cases *i* and *ii* the flaw is repaired by adding an ordering constraint to π, which further constrains π, thus $cost(\pi) \leq cost(\pi')$. Case *iii* results in a new causal link and an ordering constraint, and is therefore the same as cases *i* and *ii*. In case *iv*, the action's optimization model is added to π, but since the cost function of the action must be non-negative under its constraints, $cost(\pi')$ cannot be less than $cost(\pi)$. By applying this argument inductively on the complete branch-and-bound subtree grown from π, we get $cost(\pi) \leq cost(\bar{\pi})$ for any completion $\bar{\pi}$ of π. □

5.3.2 Domain Independent Heuristic Cost Estimation

Although $cost(\pi)$ provides a reasonable lower bound for π, the bound is only computed over actions in the plan. The bound can be strengthened by also reasoning over actions that are needed to complete the plan. We present an extension of h_{max} [61], called h^{cost}_{max}, which estimates the cost of achieving the open conditions of a plan π in a similar manner to VHPOP [166]. The extension is that instead of using action cost, we use the (precomputed) value of the minimized optimization model of an action (M_a).

$$h^{cost}_{max}(\omega, \pi) = \begin{cases} 0 & \text{if } \omega \subseteq \mathit{effs}_\pi, \text{ else} \\ \min_{\{a \in \mathcal{A} \setminus A | \mu \in \mathit{eff}_a\}} \{M_a + h^{cost}_{max}(\mathit{pre}_a, \pi)\} & \text{if } \omega = \{\mu\}, \text{ else} \\ \max_{\mu \in \omega} \{h^{cost}_{max}(\{\mu\}, \pi)\} & \text{if } |\omega| > 1, \end{cases} \tag{5.13}$$

where ω is a partial state variable assignment, μ is a single state variable assignment $v \mapsto d$, and $\mathit{effs}_\pi = \bigcup_{a \in A} \mathit{eff}_a$. The heuristic takes the max over the estimated cost of achieving the elements in the given assignment ω. The cost is zero if the elements are already in π, otherwise the minimum cost of achieving each element is computed by finding the cheapest way of bringing that element into the plan. A comparison of h_{max} to costed-RPG style heuristics (such as that of POPF) can be found in [40]. Figure 5.4 shows an example of the h^{cost}_{max} heuristic in action. Computing h^{cost}_{max} can be done with

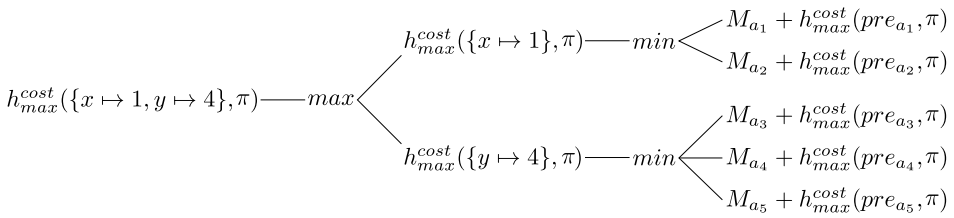


Figure 5.4: In this example, the h^{cost}_{max} heuristic is estimating the cost of achieving the state variable mapping $\{x \mapsto 1, y \mapsto 4\}$ in the plan π, where $a_1, \ldots, a_5$ are actions with $\{x \mapsto 1\} \in (\mathit{eff}_{a_1} \cap \mathit{eff}_{a_2})$ and $\{y \mapsto 4\} \in (\mathit{eff}_{a_3} \cap \mathit{eff}_{a_4} \cap \mathit{eff}_{a_5})$.

a dynamic programming approach that stores the cost of achieving individual state variable mappings.

We now prove that the h^{cost}_{max} heuristic combined with the lower bound $cost(\pi)$ never overestimate the cost of a partial plan.

Proposition 5.2. *Given any valid partial plan $\pi = \langle A, C, O, M \rangle$ where $M_a \geq 0$, $\forall a \in A$, $cost(\pi) + h^{cost}_{max}(open(\pi), \pi) \leq cost(\bar{\pi})$ for any completion $\bar{\pi}$ of π.*

Proof. We have $h^{cost}_{max}(\omega, \pi) = \sum_{a \in R} M_a$, where R is the set of actions not currently in π ($R = \mathcal{A} \setminus A$) that are required to resolve ω and among such sets has the minimum cost. Thus, any completion $\bar{\pi}$ of π as described in Proposition 5.1 must at least increase $cost(\pi)$ by $h^{cost}_{max}(\omega, \pi) = \sum_{a \in R} M_a$. □

It is possible to extend more recent work in admissible heuristics for cost-optimal planning with the techniques and heuristics in, e.g, [59, 60, 64, 65, 78] in the same way to produce even more accurate admissible estimates. We leave this for future work.

5.3.3 LTOP Model

We describe the main components of the LTOP model in this section. For a detailed overview, we refer to Appendix B. Our model of the NCLSFRP shares a number of similarities with the PDDL model, using most of the same types of actions: `sail`, `sail-on-service`, `sail-equipment`, `phase-out` and `phase-in`. We model all of these actions in the same way, with the repositioning of a vessel starting and ending with a phase-out and phase-in action, respectively, and sailing, SOS and sail equipment actions being ordered in between.

As in the PDDL model, we keep track of the state of the vessel, however in LTOP we use state variables. We only allow a vessel to begin using sailing, SOS, and sailing with equipment activities once it is in a transit state, meaning it has used a phase-out action. We also only consider the vessel as having completed its repositioning once it enters its goal state, which is accomplished through a phase-in. We also store the location of each vessel in a state variable, and use this to determine which actions can be applied for that vessel. Note that the state variables in LTOP make the mutual

exclusivity of the vessel state more explicit than in the PDDL model. For example, a state variable `vessel-at(ves)` $\in P$, where `ves` is a vessel and P is the set of ports, can only take a single value by its definition. In PDDL, there are predicates for each port and vessel pair, e.g., `(vessel-at ves p1)` $\in \mathbb{B}$ and `(vessel-at ves p2)` $\in \mathbb{B}$ meaning planners must spend time determining that `ves` can never be in both `p1` and `p2` at the same time, whereas in LTOP this is known from the start.

The initial state of the model puts all vessels in a non-phased out state, meaning they may not yet perform any repositioning activities. The vessels are only assigned a location once they have phased out. The goal state of the model requires that every vessel has performed a phase-in.

Hotel period

The LTOP model further differentiates itself from the PDDL model, in which we needed an envelope action to model hotel costs, in that in LTOP, we can keep track of the hotel costs directly with optimization variables. We associate the variables h_v^B and h_v^E with the hotel begin and end time of vessel v, respectively. To update the hotel costs, we associate the following constraints with every action $a \in \mathcal{A}$:

$$\begin{aligned} h_v^B &\leq x_a^B \qquad \forall v \in V \\ h_v^E &\geq x_a^E \qquad \forall v \in V, \end{aligned}$$

where V is the set of vessels, and x_a^B and x_a^E are the begin and end time of action a as previously defined. In this way, the lower and upper bound of the hotel period is updated every time an action is added to the current plan. We add the hotel cost to the objective in the phase-in action because LTOP is a partial-order planner, meaning it plans backwards from the goal state. Phase-in actions are therefore the first actions added to the plan, as they satisfy the goal state. This ensures the hotel cost is taken into account throughout the entire search.

5.3.4 Fleet Repositioning Specific Heuristics

We also implemented two domain-specific heuristics in order to better solve the NCLS-FRP. First, we modified the branching scheme of LTOP in order to avoid multiple `sail` actions in a row, observing that it will never be cheaper to sail through an intermediate port than to directly sail between two ports. This is straightforward to implement, requiring only for LTOP to check if adding a `sail` action to a partial plan would result in two sailings in a row. We could also implement this through a state variable that prevents multiple sailings like the `can-sail` fact in the PDDL model. However, LTOP has a hard time coping with such a scenario because multiple sail actions can be added, which then result in a plan infeasibility. Many of these infeasible plans must be explored, meaning the gain from adding such a state variable is limited. We blame this mainly on LTOP's simple planning heuristics. Future work could focus on strength-

Inst.	LTOP Actions	LTOP				POPF (Optimal)		POPF (Satisficing)	
		DLH	DL	LH	L	Forwards	Reversed	Standard	
AC3_1_0	99	**1.1**	1.1	1.1	1.1	0.7	1.4	0.4	(0.0)
AC3_2_0	182	51.0	51.5	**50.4**	53.5	-	809.6	32.5	(0.0)
AC3_3_0	252	**188.3**	196.8	193.3	202.0	-	-	1105.1	(0.0)
AC3_1_1e	219	3.9	3.9	5.2	5.3	**3.3**	4.0	1.7	(0.0)
AC3_2_2ce	419	**15.2**	25.2	55.2	126.8	-	-	1550.6	(0.3)
AC3_3_2c	447	**203.2**	362.2	2979.7	3715.8	-	-	399.2	(0.2)
AC3_3_2e	590	**217.1**	263.0	1453.1	2092.8	-	-	291.5	(1.3)
AC3_3_2ce1	590	**218.2**	260.8	1451.6	2068.4	-	-	303.9	(1.3)
AC3_3_2ce2	590	**192.4**	216.0	2624.1	3094.3	-	-	1464.2	(1.6)
AC3_3_2ce3	596	**516.9**	685.5	2959.1	-	-	-	348.0	(1.1)
AC3_3_3	477	**80.0**	102.4	735.0	1140.8	-	-	1975.5	(2.3)

Table 5.3: CPU times for LTOP and POPF solving instances from the NCLSFRP dataset to optimality, as well as optimality gaps for POPF when satisficing. We configure LTOP to use the domain-specific heuristics (D), the h^{cost}_{max} heuristic (H), and the LP lower bound heuristic (L),

ening these heuristics in order to handle the above situation in a domain-independent way.

Our second heuristic is able to complete a vessel's repositioning once the vessel is assigned a `sail-on-service` action. Since the starting time of the `sail-on-service` is fixed, and so are the times of the phase-outs, the optimal completion to the plan can be computed by simply looping over the vessel's allowed phase-out ports and choosing the one with the lowest cost. We add a `phase-out` action to the plan along with a `sail` action, if necessary.

5.3.5 LTOP Computational Evaluation

We implemented LTOP in C++ and evaluate it on the NCLSFRP dataset. Table 5.3 shows the CPU times in seconds of using LTOP with varying heuristics to solve the NCLSFRP dataset to optimality with a one hour timeout. Additionally, the table provides the number of grounded LTOP actions in each instance. We ran LTOP with domain-specific heuristics (D), the h^{cost}_{max} heuristic (H) for computing lower bounds and the LP lower bound heuristic (L). For comparison purposes, we show the optimal solution times of POPF using both the forward and reverse NCLSFRP PDDL domain, as well as the best performing satisficing configuration of POPF. For the POPF's satisficing solutions, we show the solution's optimality gap in parenthesis. The lowest CPU time on each instance found by an optimal planner is highlighted in bold.

LTOP is successful in solving all of the NCLSFRP instances to optimality, even without domain-specific heuristics, a feat POPF is unable to match. The DLH heuristic combination is most effective at solving NCLSFRP instances, achieving the best performance on 9 out of 11 instances, and near-best performance on the remaining 2 instances. On instances larger than AC3_3_2c, domain-specific heuristics are critical to finding a solution quickly. Overall, using domain-specific heuristics result in an over 14 times speed up than using just the LP and h^{cost}_{max}. The h^{cost}_{max} heuristic provides small

speed improvements over not using it when domain-specific heuristics are involved, resulting in DLH having a 16% better solution time on average than DL. However, in the domain-independent case, h^{cost}_{max} is critical for finding an optimal solution to all instances and results in a 51% speed up on all instances for LH over L.

LTOP, using the DLH heuristic combination, is also able to find an optimal solution in less time than POPF takes to find a satisficing solution on 7 instances. POPF is actually able to find an optimal answer in the satisficing configuration faster than LTOP in three cases: on instances AC3_1_0, AC3_2_0, and AC3_1_1e. However, since satisficing POPF cannot prove the optimality of the solution it finds we do not highlight this in the table. Nonetheless, it indicates that POPF excels mainly at the small instances in the dataset, while LTOP is capable of handling even larger instances.

5.4 A Mixed-integer Programming Model of Fleet Repositioning

Despite the success of LTOP in solving NCLSFRP instances, the question remains as to how it performs versus more traditional combinatorial optimization techniques. To find out, we create a mixed-integer programming (MIP) model of the NCLSFRP that considers the activities that a vessel may undertake and connects activities based on which ones can feasibly follow one another temporally. The structure of the NCLSFRP is embedded directly into the graph of the MIP, meaning that it is unable to model general automated planning problems as in [80] and [157]. Note that, unlike LTOP, the MIP is capable of handling negative activity costs.

Since the vessel state in fleet repositioning is relatively simple, encompassing where a vessel is and whether it has begun its repositioning or not, there is not an exponential growth in the number of graph nodes as there would be in many planning problems if they were modeled using a graph.

5.4.1 Graph and MIP Description

Given a graph $G = (A, T)$, where A is the set of actions (nodes), and T is the set of transitions, with $(a, b) \in T$ iff action b may follow action a, let the decision variable $y_{a,b} \in \{0, 1\}$ indicate whether or not the transition $(a, b) \in T$ is used or not. The auxiliary variable $w_a = \sum_{(a,b)\in T} y_{a,b}$ indicates whether action a is chosen by the model, and $x^s_a, x^e_a \in \mathbb{R}^+$ are action a's start and end time, respectively. Finally, the variables h^s_v and h^e_v are the start and end time of the hotel cost period for vessel v.

Each action $a \in A$ is associated with a fixed cost, $c_a \in \mathbb{R}$, a variable (hourly) cost, $\alpha_a \in \mathbb{R}$, and a minimum and maximum action duration, d^{min}_a and d^{max}_a. The set $A^t \subseteq A$ specifies actions that must begin at a specific time, t_a. The use of a particular action may exclude the use of other actions. These exclusions are specified by $\eta : A \to 2^{|A|}$. There are also n sets of mutually exclusive actions, given by $\mu : \{1, \ldots, n\} \to 2^{|A|}$. We differentiate between phase-out and phase-in actions for each vessel using the sets

$A_v^{po}, A_v^{pi} \subseteq A^t$, respectively, and let $A' = A \setminus \cup_{v \in V}(A_v^{po} \cup A_v^{pi})$ be all actions that are not related to the phase-out or phase-in. Finally, let $c_v^H \in \mathbb{R}^+$ represent each vessel's hourly hotel cost.

There are several "big-Ms" in the model, which are constants used in MIP models to enforce logical constraints. The upper bound on the difference between the end and start of two actions is given by $M_{a,b}^y$. The upper bound on the start of a vessel's hotel period, and the lower bound on the end of the vessel's hotel period, are given by M_v^s and m_v^e respectively.

Parameters

We now summarize the parameters and variables used in our MIP model for easy reference. The model uses the following parameters:

$G = (A, T)$	Graph of nodes A (actions) and arcs T (transitions).
V	Set of vessels.
A^{po_v}	Set of phase-out actions for vessel $v \in V$.
A^{pi_v}	Set of phase-in actions for vessel $v \in V$.
c_a	Fixed cost of action a.
α_a	Variable cost of action a.
c_v^H	Hotel cost of vessel $v \in V$ per hour.
d_a^{min}, d_a^{max}	Minimum and maximum duration of action a.
$M_{a,b}^y$	Maximum time difference between the start of actions a and b.
M_v^s, m_v^e	Upper and lower bound of the start and end of vessel $v \in V$'s hotel period.
$\mu(a)$	Set of actions that are mutually exclusive with a.
$\eta(a)$	Set of actions that are excluded by a (but not necessarily mutually exclusive).

Variables

The model uses the following variables:

$y_{a,b} \in \{0, 1\}$	Indicates whether transition $(a, b) \in T$ is used.
$w_a = \sum_{(a,b) \in T} y_{a,b}$	Auxiliary variable indicating whether action a is used or not.
$x_a^s, x_a^e \in \mathbb{R}^+$	The start and end time of action $a \in A$, respectively.

Objective and constraints

The objective and constraints are as follows:

$$\min \sum_{a \in A} (c_a w_a + \alpha_a (x_a^e - x_a^s)) + \sum_{v \in V} c_v^H (h_v^e - h_v^s) \tag{5.14}$$

$$\text{s.t.} \sum_{\{(a,b)\in T \mid b\in A\setminus A_v^{po}\}} y_{a,b} = 1 \qquad \forall v \in V, a \in A_v^{po} \tag{5.15}$$

$$\sum_{(a,b)\in T} y_{a,b} = \sum_{(b,c)\in T} y_{b,c} \qquad \forall b \in A' \tag{5.16}$$

$$\sum_{(a,b)\in T} y_{a,b} \leq 1 \qquad \forall b \in A' \tag{5.17}$$

$$x_a^e - x_b^s \leq M_{a,b}^y (1 - y_{a,b}) \qquad \forall (a,b) \in T \tag{5.18}$$

$$x_a^s \leq x_a^e \qquad \forall a \in A \tag{5.19}$$

$$d_a^{min} w_a \leq x_a^e - x_a^s \leq d_a^{max} w_a \qquad \forall a \in A \tag{5.20}$$

$$x_a^s = t_a w_a \qquad \forall a \in A^t \tag{5.21}$$

$$h_v^s + M_v^s w_a \leq M_v^s + t_a \qquad \forall a \in \bigcup_{v\in V} A_v^{po} \tag{5.22}$$

$$h_v^e + m_v^e w_a \geq m_v^e + t_a \qquad \forall a \in \bigcup_{v\in V} A_v^{pi} \tag{5.23}$$

$$\sum_{a\in\mu(i)} w_a \leq 1 \qquad \text{for } i = 1, 2, \ldots n \tag{5.24}$$

$$|\eta(a)| w_a + \sum_{b\in\eta(b)} w_b \leq |\eta(a)| \qquad \forall a \in A \tag{5.25}$$

The objective, (5.14), sums the fixed and variable costs of each action that is used along with the hotel cost for each vessel. The single unit flow (i.e., node disjoint) structure of the graph is enforced in constraints (5.15) start the flow of vessels through the graph, ensuring that every vessel transitions out of its phase-out. Constraints (5.16) are flow balance constraints that ensure vessels traverse a single path through the graph. Constraints (5.17) limit the incoming number of vessels to any activity to one, meaning that only a single vessel may undertake any particular activity. Constraints (5.18) enforce the ordering of transitions between actions, preventing the end of one action from coming after the start of another if the edge between them is turned on. Action start and end times are ordered by (5.19), and the duration of each action is limited by (5.20). Actions with fixed start times are bound to this time in (5.21). Constraints (5.22) and (5.23) connect the hotel start and end times to the time of the first and last action, respectively. Note that the objective forces h_v^s and h_v^e as close together as possible, and that the big/little-Ms are required because each action has a different start time. The mutual exclusivity of certain sets of actions is enforced in constraints (5.24). Finally, constraints (5.25) prevents actions from being included in the plan if they are excluded by an action that was chosen. These constraints are primarily used to ensure the block phase in structure necessary to have a weekly temporal spacing of vessels. When a phase-in is chosen for a vessel, phase-ins that could not possibly be used with it are disabled. For example, in a 3 vessel problem, any phase in more than 2 weeks later from a particular phase-in is disabled if a vessel uses that phase-in.

Inst.	MIP	LTOP				POPF (Optimal)		POPF (Satisficing)	
		DLH	DL	LH	L	Forwards	Reversed	Standard	
AC3_1_0	**0.4**	1.1	1.1	1.1	1.1	0.7	1.4	0.4	(0.0)
AC3_2_0	**9.3**	51.0	51.5	50.4	53.5	-	809.6	32.5	(0.0)
AC3_3_0	**23.0**	188.3	196.8	193.3	202.0	-	-	1105.1	(0.0)
AC3_1_1e	**3.8**	3.9	3.9	5.2	5.3	3.3	4.0	1.7	(0.0)
AC3_2_2ce	27.7	**15.2**	25.2	55.2	126.8	-	-	1550.6	(0.3)
AC3_3_2c	250.5	**203.2**	362.2	2979.7	3715.8	-	-	399.2	(0.2)
AC3_3_2e	228.8	**217.1**	263.0	1453.1	2092.8	-	-	291.5	(1.3)
AC3_3_2ce1	312.2	**218.2**	260.8	1451.6	2068.4	-	-	303.9	(1.3)
AC3_3_2ce2	252.6	**192.4**	216.0	2624.1	3094.3	-	-	1464.2	(1.6)
AC3_3_2ce3	706.5	**516.9**	685.5	2959.1	-	-	-	348.0	(1.1)
AC3_3_3	148.3	**80.0**	102.4	735.0	1140.8	-	-	1975.5	(2.3)

Table 5.6: CPU times for the MIP model in CPLEX 12.3 versus LTOP using domain-specific heuristics (D), the h^{cost}_{max} heuristic (H), and the LP lower bound heuristic (L), and POPF.

5.4.2 MIP Model Computational Evaluation in CPLEX

Table 5.6 provides the CPU times of solving the MIP model in CPLEX 12.3 to optimality. We also show the results from LTOP and POPF which are explained in detail in Sections 5.3.5 and 5.2.4, respectively.

The MIP outperforms LTOP on AC3_1_0 through AC3_1_1e, the smallest instances in our dataset. However, once the instances begin growing in size with AC3_2_2ce, LTOP requires only 75% of the time of the MIP with the DLH heuristics. The MIP easily outperforms LTOP with only domain-independent heuristics (LH and L), but this is not surprising considering that the MIP is able to take our domain-specific heuristics into account through its graph construction. The instance that most realistically represents the scenario our industrial collaborator faced is AC3_3_3, which LTOP is able to solve in slightly over half the time of the MIP. Overall, the MIP requires an average time of 178 seconds versus only 153 seconds for LTOP on our dataset. We suspect this performance to be due to the multitude of big-"M" constraints, which are difficult for MIP solvers to deal with. Planning techniques are better able to handle these sorts of logical constraints and manage the state surrounding a vessel throughout its repositioning.

5.5 A Constraint Programming Model of Fleet Repositioning

In addition to our PDDL, LTOP and MIP models, we present a novel CP model for the NCLSFRP, due to the success of CP in solving scheduling problems in the literature. This CP model was first presented in [83]. Our model exploits the lack of chaining of SOS and SE activities, which means it is significantly less extensible than the PDDL, LTOP or MIP models, but is able to find solutions in less CPU time.

5.5.1 Model Description

Let O_v be the set of possible phase-outs for the vessel v, and let P be the be the set of possible phase-in ports for the new service. Each phase-out is associated with a particular port and time, which we will specify later. The decision variable $\rho \in P$ is the phase-in port for all vessels. The decision variables $w_v \in \{1, \ldots, W\}$ represent the phase-in week for each vessel v, where W is the number of weeks considered in the problem. For each vessel v we also define a decision variable $q_v \in O_v$ specifying the phase-out action (port and time) used for that vessel.

For each vessel v and phase-out action o, the function $t(v, o)$ specifies the phase-out time for that action. Similarly, $t(p, w)$ specifies the phase-in time for a vessel phasing in at port p in week w. The function $C(v, o, p, w)$ specifies the cost for vessel v using the phase-out action o, and phasing in at port p in week w, with -1 as a flag that indicates vessel v cannot phase in at port p in week w if it phased out using action q (for example, if action q starts too late for vessel v to reach port p in time). The dependent variable c_v specifies the cost for vessel v when the vessel sails directly from the phase-out port to the phase-in port. For each vessel v, $C_H(v)$ specifies its hourly hotel cost, and h_v is the duration of the hotel cost time period (from the phase-out to the phase-in).

We split each SOS opportunity into several *SOS actions*, where each SOS action represents starting the SOS at a different port on the SOS service. SOS opportunities save money by allowing vessels to sail for free between two ports, however a cost for transshipping cargo at each side of the SOS is incurred. Let S be the set of available SOS actions and S' be the set of SOS opportunities. The decision variable $s_v \in S$ specifies the SOS action used for each vessel v, with 0 being a flag indicating that vessel v does not use an SOS action. For each SOS action $s \in S$, the function $y : S \to S'$ specifies which SOS opportunity each SOS action belongs to, with $y(s) = 0$ being a flag that specifies that the vessel is not using any SOS opportunity.

In order to use an SOS opportunity, a vessel must sail to the starting port of the SOS opportunity before a deadline, and after using the SOS, it sails from the end port at a pre-determined time to the phase-in port. The function $C^{to}(v, s, o)$ specifies the cost of vessel v using SOS action s, phasing out at phase-out o going to the SOS action, and $C^{from}(v, s, p, w)$ is the cost of vessel v to sail from SOS action s to phase in port p in week w, with -1 as a flag that indicates that this combination of vessel, SOS action, phase-in port and week is infeasible. The dependent variables σ_v^{to} and σ_v^{from} specify the SOS costs for vessel v for sailing to and from the SOS, respectively. The function $A(v, s)$ specifies the cost savings of vessel v using SOS action s, and the dependent variable σ_v^{dur} specifies the SOS cost savings for vessel v on the SOS.

Let Q be the set of *sail-equipment* (SE) opportunities, which are pairs of ports in which one port has an excess of a type of equipment, e.g., empty containers, and the other port has a deficit. Since we do not include a detailed view of cargo flows in this version of the LSFRP, SE opportunities save money by allowing vessels to sail for free between two ports as long as the vessel sails at its slowest speed. The cost then increases linearly as the vessel sails faster. Note that although ship's bunker

consumption functions are roughly cubic, we do not use a piecewise linear function to approximate them, as a single line allows the model to come close enough to the actual sailing costs. Let the decision variable $e_v \in E$ be the SE opportunity undertaken by vessel v, with $e_v = 0$ indicating that no SE opportunity is used. Let the decision variables d_v^{to}, d_v^{dur} and d_v^{from} be the duration of vessel v sailing to, during, and from an SE opportunity.

The functions $C^{to}(v, e, o)$, $C^{dur}(v, e)$ and $C^{from}(v, e, p, w)$ specify the fixed costs of sailing to, utilizing, and then sailing from SE opportunity e, where v is the vessel, o is the phase-out port/time, p is the phase-in port and w is the phase-in week. Together with the constant α_v, which is the variable sailing cost per hour of vessel v, the hourly cost of sailing can be computed. This is necessary since SE opportunities are not fixed in time and, thus, must be scheduled. Let the dependent variables $\lambda_v^{to}, \lambda_v^{dur}$ and λ_v^{from} be the fixed costs sailing to, on and from an SE opportunity. Additionally, let $\Delta_{min}^{to}(v, e, o)$, $\Delta_{min}^{dur}(v, e)$ and $\Delta_{min}^{from}(v, e, p, w)$ be the minimum sailing time of v before, during and after the SE opportunity and $\Delta_{max}^{to}(v, e, o)$, $\Delta_{max}^{dur}(v, e)$ and $\Delta_{max}^{from}(v, e, p, w)$ be the maximum sailing time of v before, during and after the SE opportunity.

In this version of the LSFRP, the chaining of SOS and SE opportunities is not allowed, meaning each vessel has the choice of either sailing directly from the phase-out to the phase-in, undertaking an SOS, or performing an SE. The decision variable $r_v \in \{\texttt{SOS}, \texttt{SE}, \texttt{SAIL}\}$ specifies the type of repositioning for each vessel v, where v utilizes an SOS opportunity, SE opportunity, or sails directly from the phase-out to the phase-in, respectively.

Parameters

We summarize the parameters used in the model in the following table.

V	Set of vessels.
O_v	Phase-out actions for vessel $v \in V$.
P	Set of phase-in ports.
W	Number of weeks in the problem.
$t(v, o)$	Phase-out time of phase-out activity $o \in O_v$ for vessel $v \in V$.
$t(p, w)$	Phase-in time to join the goal service at port $p \in P$ in week $w \leq W$.
$C(v, o, p, w)$	Cost for vessel $v \in V$ to use phase-out action $o \in O_v$ and phase-in at $p \in P$ in week $w \leq W$.
$C_H(v)$	Hourly hotel cost for vessel $v \in V$.
S'	Set of SOS opportunities, each consisting of multiple SOS actions representing sailings from an SOS start port to an end port.
S	Set of SOS actions.
$y(s)$	Maps each SOS action to the SOS opportunity it belongs to.
$C^{to}(v, s, o)$	Cost of sailing vessel $v \in V$ from phase-out $o \in O_v$ to SOS $s \in S$.
$C^{from}(v, s, p, w)$	Cost of sailing vessel $v \in V$ SOS $s \in S$ to phase in at port $p \in P$ in week $w \leq W$.

Q	Set of SE opportunities.
$C^{to}(v, e, o)$	Fixed cost of sailing vessel $v \in V$ to SE $e \in Q$ from phase-out $o \in O_v$.
$C^{dur}(v, e)$	Fixed cost of sailing vessel $v \in V$ on SE $e \in Q$.
$C^{from}(v, e, p, w)$	Fixed cost of sailing $v \in V$ from SE $e \in Q$ to phase in at port $p \in P$ in week $w \leq W$.
α_v	Variable sailing cost coefficient for vessel $v \in V$.
$\Delta^{to}_{min}(v, e, o)$	Minimum sailing time for vessel $v \in V$ to SE $e \in Q$ from phase-out $o \in O_v$.
$\Delta^{to}_{max}(v, e, o)$	Maximum sailing time for vessel $v \in V$ to SE $e \in Q$ from phase-out $o \in O_v$.
$\Delta^{dur}_{min}(v, e)$	Minimum sailing time for vessel $v \in V$ on SE $e \in Q$.
$\Delta^{dur}_{max}(v, e)$	Maximum sailing time for vessel $v \in V$ on SE $e \in Q$.
$\Delta^{from}_{min}(v, e, p, w)$	Minimum sailing time for vessel $v \in V$ from SE $e \in Q$ to phase in at port $p \in P$ in week $w \leq W$.
$\Delta^{from}_{max}(v, e, p, w)$	Maximum sailing time for vessel $v \in V$ from SE $e \in Q$ to phase in at port $p \in P$ in week $w \leq W$.

Variables

We summarize the decision variables of the model in the following table. We make a distinction between decision variables and dependent variables, in that dependent variables are assigned values based on the values of decision variables. The solver need not branch on any dependent variables; it is sufficient to branch solely on the decision variables in order to find an optimal solution. The dependent variables are only present in order to perform the objective computation.

$\rho \in P$	Phase-in port used by all vessels.
$d^{to}_v, d^{dur}_v, d^{from}_v \in \mathbb{R}^+_0$	Sailing duration to, during, and from an SE for vessel $v \in V$.
$e_v \in E$	SE action used by vessel $v \in V$ (if any).
$h_v \in \mathbb{R}^+_0$	Hotel period duration for vessel $v \in V$.
$q_v \in O_v$	Phase-out action (which is a port and a time) for vessel $v \in V$.
$r_v \in \{\texttt{SOS}, \texttt{SE}, \texttt{SAIL}\}$	Type of repositioning for vessel $v \in V$.
$s_v \in S$	SOS action used by vessel $v \in V$ (if any).
$w_v \in \{1, \ldots, W\}$	Phase-in week for each vessel $v \in V$.
$\lambda^{to}_v, \lambda^{dur}_v, \lambda^{from}_v \in \mathbb{R}^+_0$	Dependent variable specifying the costs of sailing to, during, and from an SE opportunity for vessel $v \in V$.
$\sigma^{dur}_v \in \mathbb{R}^+_0$	Dependent variable containing the cost savings for vessel $v \in V$ over the duration of the SOS.
$\sigma^{to}_v(\sigma^{from}_v) \in \mathbb{R}^+_0$	Dependent variable containing the cost of vessel $v \in V$ to sail to (from) the phase-out to an SOS.
$c_v \in \mathbb{R}^+_0$	Dependent variable specifying the cost for vessel $v \in V$ to sail directly from the phase-out to the phase-in.

Objective and Constraints

$$\min \sum_{v \in V} \big(C_H(v) \left(t(\rho, w_v) - t(v, q_v) \right) + c_v + \sigma_v^{from} + \sigma_v^{dur} + \sigma_v^{to} + \lambda_v^{to} + \lambda_v^{dur} + \lambda_v^{from} + \alpha_v (d_v^{to} + d_v^{dur} + d_v^{from}) \big) \tag{5.26}$$

$$\text{s. t.} \quad \texttt{alldifferent}(w_v), v \in V \tag{5.27}$$

$$\max_{v \in V} w_v - \min_{v \in V} w_v = |V| - 1 \tag{5.28}$$

$$\texttt{alldifferent\_except\_0}(s_v), v \in V \tag{5.29}$$

$$\texttt{alldifferent\_except\_0}(y(s_v)), v \in V \tag{5.30}$$

$$r_v = \texttt{SAIL} \rightarrow c_v = C(v, q_v, \rho, w_v), \quad \forall v \in V \tag{5.31}$$

$$r_v = \texttt{SAIL} \rightarrow \big(s_v = 0 \wedge y(s_v) = 0 \wedge \sigma_v^{dur} = 0 \wedge \sigma_v^{from} = 0 \wedge \sigma_v^{to} = 0 \wedge e_v = 0 \wedge \lambda_v^{to} = 0 \wedge \lambda_v^{dur} = 0 \wedge \lambda_v^{from} = 0 \big), \quad \forall v \in V \tag{5.32}$$

$$r_v = \texttt{SOS} \rightarrow s_v > 0 \wedge y(s_v) > 0 \wedge \sigma_v^{dur} = -A(v, s_v) \wedge \sigma_v^{from} = C^{from}(v, s_v, \rho, w_v) \wedge \sigma_v^{to} = C^{to}(v, s_v, q_v), \quad \forall v \in V \tag{5.33}$$

$$r_v = \texttt{SOS} \rightarrow c_v = 0 \wedge e_v = 0 \wedge \lambda_v^{to} = 0 \wedge \lambda_v^{dur} = 0 \wedge \lambda_v^{from} = 0, \quad \forall v \in V \tag{5.34}$$

$$s_v > 0 \vee y(s_v) > 0 \rightarrow r_v = \texttt{SOS}, \quad \forall v \in V \tag{5.35}$$

$$\texttt{alldifferent\_except\_0}(e_v), \quad \forall v \in V \tag{5.36}$$

$$r_v = \texttt{SE} \rightarrow e_v > 0 \wedge \lambda_v^{to} = C^{to}(v, e_v, q_v) \wedge \lambda_v^{dur} = C^{dur}(v, e_v) \wedge \lambda_v^{from} = C^{from}(v, e_v, \rho, w_v), \forall v \in V \tag{5.37}$$

$$r_v = \texttt{SE} \rightarrow s_v = 0 \wedge y(s_v) = 0 \wedge \sigma_v^{dur} = 0 \wedge c_v = 0 \wedge \sigma_v^{from} = 0 \wedge \sigma_v^{to} = 0, \forall v \in V \tag{5.38}$$

$$e_v > 0 \rightarrow r_v = \texttt{SE} \tag{5.39}$$

$$\Delta_{min}^{to}(v, e_v, q_v) \leq d_v^{to} \leq \Delta_{max}^{to}(v, e_v, q_v), \quad \forall v \in V \tag{5.40}$$

$$\Delta_{min}^{dur}(v, e_v) \leq d_v^{dur} \leq \Delta_{max}^{dur}(v, e_v), \quad \forall v \in V \tag{5.41}$$

$$\Delta_{min}^{from}(v, e_v, \rho, w_v) \leq d_v^{from} \leq \Delta_{max}^{from}(v, e_v, \rho, w_v), \quad \forall v \in V \tag{5.42}$$

$$\sigma_v^{to}, \sigma_v^{from}, c_v \geq 0, \quad \forall v \in V \tag{5.43}$$

The objective function (5.26) minimises the sum of the hotel costs and repositioning action costs minus the cost savings for SOS actions for the set of vessels. Constraints (5.27) and (5.28) specify that the vessels must all phase in to the new service on different, successive weeks. Constraints (5.29) and (5.30) specify that all vessels using SOS actions must use different actions and action types. `alldifferent_except_0` is a global constraint that requires all elements of an array of variables to be different, except those that have the value 0. This allows us to enforce the all different constraint only when an SOS action is actually being used by a vessel.

Constraints (5.31) and (5.32) set the costs for a vessel if it uses a `SAIL` repositioning, and ensures that the SOS/SE actions and costs are set to 0, as they are not being used.

Constraints (5.33) and (5.34) specify that if vessel v uses an SOS (`SOS`) repositioning action s_v, then its repositioning cost is equal to the costs for sailing to and from that SOS action based on the phase-out action, phase-in port and week, minus the cost savings $A(v, s_v)$ for that SOS action. In addition, the normal repositioning cost c_v and the sail equipment action for that vessel are set to 0. We also add redundant constraints (5.35) to reinforce that the repositioning type be set correctly when an SOS is chosen.

In constraints (5.36) we ensure that no two vessels choose the same SE action (unless they choose no SE action), and constraints (5.37) and (5.38) bind the costs of the sail equipment action to the dependent variables if an SE is chosen, as well as set the costs of a direct sailing and SOS opportunities for each vessel to 0. The redundant constraints (5.39) ensure that the repositioning type of vessel v is correctly set if an SE action is chosen. The minimum and maximum durations of the parts of the SE (sailing to the SE from the phase-out, the SE itself, and sailing from the SE to the phase-in) are set in constraints (5.40), (5.41) and (5.42). Constraint (5.43) requires that all SOS actions and phase-out/phase-in combinations must be valid for each vessel (i.e., transitions with -1 costs must not be used).

5.5.2 CP Model Computational Evaluation in G12

We formulated the above CP model in the MiniZinc 1.6 modeling language [104, 105] and solved it to optimality using the CPX solver in G12 2.0 [45, 162]. Note that in our CP model for MiniZinc we had to add constraints on the maximum duration of SE actions, as well as a constraint on the maximum sum of the objective, in order to prevent integer overflows. These constraints do not cut off any valid solutions from the search tree. Since MiniZinc does not support floating point objective values, the MiniZinc model is an extremely close approximation of the true objective. The objectives it computes can, at times, be several cents away from the true objective. In no case does this rounding result in a non-optimal plan being found.

We also used several *search annotations* within MiniZinc to help guide the solver to a solution. Search annotations are a way of providing advice to solvers about which branching decisions to take and which values in a variable's domain to explore first.

The first annotation we add is to branch on the type of repositioning, r_v, before other variables. The annotation further forces the solver to first try to find a SOS option for each vessel, then moves on to SE options, and finally checks SAIL options. This search order was the most efficient for the most complex models that include both SOS and SE opportunities, since SE constraints are more complex than SOS constraints, so searching SE options first is more time consuming for models that contain both SOS and SE opportunities.

For instances with SE opportunities, we also add a search annotation to branch on the SE opportunity, e_v, using the *indomain_split* functionality of MiniZinc, which excludes the upper half of a variable's domain. Both annotations use a first failure strategy, meaning the variable the solver branches on is the one with the smallest domain.

Inst	LTOP-DLH	MIP	CP	CP–A	CP–AO	CP–R	CP–AOR
AC3_1_0	1.1	0.4	0.1	0.2	0.1	0.2	0.1
AC3_2_0	51.0	9.3	0.3	0.3	0.3	0.3	0.3
AC3_3_0	188.3	23.0	0.6	0.6	0.6	0.6	0.6
AC3_1_1e	3.9	3.8	0.7	0.7	0.7	0.8	0.4
AC3_2_2ce	15.2	27.7	-	83.9	25.5	-	11.3
AC3_3_2c	203.2	250.5	6.0	3.1	3.1	6.2	3.0
AC3_3_2e	217.1	228.8	1731.0	15.8	16.2	1742.8	13.2
AC3_3_2ce1	218.2	312.2	32.6	23.0	16.7	31.5	13.9
AC3_3_2ce2	192.4	252.6	64.4	25.4	23.3	63.7	17.1
AC3_3_2ce3	516.9	706.5	-	-	695.4	-	470.3
AC3_3_3	80.0	148.3	18.1	11.0	11.0	18.6	11.2

Table 5.9: CPU times in seconds to optimality for the CP model with/without annotations (A), repositioning type order (O), and redundant constraints (R) vs. LTOP and the MIP.

Table 5.9 compares the run times of the CP model against the MIP model and LTOP, all of which find optimal solutions. We run the CP model without any annotations or redundant constraints, with search annotations using a search order of SAIL/SOS/SE (CP–A), with search annotations and the SOS/SE/SAIL ordering (CP–AO), with only the redundant constraints and using the solver's default search (CP–R), and using the SOS/SE/SAIL ordering with redundant constraints (CP–AOR). CP–AOR is faster than the MIP on all instances, often by an order of magnitude. The main challenge for the CP model are instances with equipment, just as for LTOP and the MIP. The CP model times out on two instances for CP and CP–R, as well as one for CP-A, whereas the MIP and LTOP solve all instances. Of particular note is that the CP model is able to solve AC3_3_3, which is the instance that most closely models the actual scenario faced by our collaborator, in only 10 seconds. Such a quick solution time allows for interaction and feedback with a repositioning coordinator within a decision support system. Note that no single improvement alone is enough to give the CP model better performance than both LTOP and the MIP.

Our CP model comes with two limitations. The first limitation is the model's flexibility. A natural extension to this model would be to allow for the chaining of SOS and SE opportunities, which is easy to do in both the LTOP and MIP models, due to automated planning's focus on actions, and our MIP model's focus on flows. However, the CP model is structured around exploiting this piece of the problem. Other natural changes, such as allowing vessels to undergo repairs, would also be difficult to implement. The second limitation is that many of the components of the CP model involve pre-computations that multiply the number of phase-out actions with the number of phase-in ports and weeks. Although the model works well on our real world instance, these pre-computations pose an issue for scaling to larger liner shipping services.

5.6 Chapter Summary

We introduced the NCLSFRP, a simplification of the overall LSFRP that captures several key difficult components of the LSFRP, including time-dependent task costs and

the coordination necessary between vessels to create a liner shipping service. We introduced four different models of the problem, using PDDL, the LTOP framework, MIP and CP. Through experimental analysis on a dataset of eleven instances based on a real-world repositioning scenario, we were able to determine the strengths and weaknesses of the various methods. We achieved the best performance across the instances using a highly domain-specific CP model, as well as impressive performance from the LTOP approach, which is not nearly as well-studied or developed as the MIP or CP methods. We showed that automated planning is capable of solving real-world combinatorial optimization problems, but that more work is needed in developing strong, domain-independent heuristics.

Chapter 6

Liner Shipping Fleet Repositioning with Cargo

Although the model of the LSFRP presented in Chapter 5 is useful for certain types of repositionings, taking into account the flows of containers through the network is important for ensuring the repositioning plans that are generated do not cause significant disruptions to the on-time delivery of containers.

To this end, we present a version of the LSFRP that includes cargo flows and has a fixed time window in which all vessels must be repositioned. In contrast with the NCLSFRP, this version of the problem contains all aspects of the LSFRP discussed in Chapter 3, including parallel SOS sailings, port inducement/omission, and a fixed deadline after which normal service on the goal service must commence.

We create a specialized graph to model the problem in which nodes represent port calls and the arcs of the graph define legal sailings between port calls. Using specialized constructions in the graph, we are able to completely model SOS opportunities, the phase-in and the phase-out without any additional mathematical constraints. Container flows are represented with a maximal multi-commodity flow over this graph.

We present two models of the LSFRP with cargo flows using this graph, using an arc flow and a path-based approach. We then model a simplified version of the LSFRP in which there are no flexible visitations, which we call the node flow model. We solve the arc flow and node flow models in CPLEX and show their scaling behavior on two real-world datasets of LSFRP instances. We solve the path-based model using a column generation procedure that provides optimal solutions on all instances in our dataset, despite only guaranteeing a lower bound. Of our two datasets, one is a confidential dataset consisting of real data from our industrial collaborator, and the other contains anonymized information from the confidential dataset. This dataset is publicly available. These datasets differ from the instances presented in Chapter 5 in that they include more components of the problem as well as cargo flows.

Furthermore, we introduce two heuristic methods for solving the LSFRP, simulated annealing (SA) [85] and late acceptance hill climbing (LAHC) [19], that both use a set of initial solution heuristics and neighborhood operators to find good solutions to the

LSFRP in less time than is generally required by the optimal approaches. We show that SA outperforms LAHC on the LSFRP with cargo flows, and that both SA and LAHC are able to scale to the largest instances of our dataset and provide solutions with low optimality gaps. In fact, SA is able to find optimal solutions on a majority of the dataset, indicating that it is well-suited to solving the LSFRP with cargo flows.

We also compare the results of our optimal and heuristic approaches to a reference instance from Maersk Line representing a real repositioning that was undertaken in 2011. We show that our methods improve the profit obtained during repositioning in the reference scenario by nearly $14 million.

6.1 Graph Construction

We model the LSFRP with cargo flows on a graph $G = (V, A)$, where V is the set of nodes and A the set of directed arcs between nodes. Each node in V represents a *visitation* of a vessel at a particular port[1], and each arc in A represents an allowed sailing between two visitations. The graph encompasses all of the activities each vessel may undertake during a fixed *repositioning period*, which is the period from the time the vessel is first allowed to leave its phase-out service until the time when normal operations must begin on the phase-in service. The path of each vessel through the graph represents the activities to be undertaken by that vessel, and we therefore require the paths to be node disjoint to prevent multiple vessels from performing the same activity. This is an important constraint because *i*) container port terminals assign timeslots to vessels, meaning there is not enough room for two vessels, and *ii*) profit from carrying cargo can only be earned a single time, removing any reason for multiple vessels to visit the same node. Note that flexible visitations, i.e., visitations without a prior fixed schedule, can be undertaken by multiple vessels, even simultaneously. For ease of modeling, we therefore replicate flexible visitations for each vessel and consider them as node disjoint. We give more details about this process (and justifications) later. We embed a number of problem constraints and objectives directly in the graph, including sailing costs, sail-on-service opportunities, cabotage restrictions, phase-in/out requirements, and canal fees, which are described in detail in the next section, followed by our MIP model over the graph.

We give a textual overview of the graph used in our model of the LSFRP with cargo flows followed by a detailed description in Section 6.1.6. The visitations in the graph are split into two disjoint sets, thus $V = V^i \cup V^f$, where V^i is the set of *inflexible* visitations, i.e, visitations associated with a specific port call time, and V^f is the set of *flexible* visitations, which are assigned a time only if a vessel performs the visitation. The set V^f contains visitations in which a vessel can pick up/deliver equipment or incremental cargo that are not on any phase-out, phase-in, or SOS service. Let S be the set of ships.

[1]We use the terms visitation and node interchangeably.

The overall structure of the graph involves four types of visitations: phase-out, phase-in, flexible, and SOS visitations. In addition to these visitations, we include a graph sink, τ, which all vessels must reach for a valid repositioning. We let $V' = V \setminus \tau$ be the set of all graph visitations excluding τ. The four types of visitations represent four disjoint sets that make up V'. We now describe the arc structure present in each of the four types of visitations.

6.1.1 Phase-out

Each ship is assigned a particular visitation, $v_s \in V'$, at which the ship $s \in S$ begins its repositioning. This visitation represents the earliest allowed phase-out time for that vessel. A visitation is then created for each subsequent port call of the ship on its phase-out slot. Each phase-out visitation is connected to the next one with an arc. Note that phase-out visitations do not connect to the phase-out visitations of other ships.

Vessels may leave phase-out nodes to sail to SOS opportunities, flexible nodes, or to a phase-in slot. Thus, arcs are created from each phase-out visitation to each phase-in visitation and SOS start visitation such that sailing between the visitations is temporally feasible (i.e., the starting time of the phase-in/SOS visitation is greater than the end time of the phase-out visitation plus the sailing time). Since flexible nodes have no fixed start and end time, arcs are created to and from all flexible nodes to all phase-outs within the same trade zone. Finally, phase-out visitations have incoming arcs from phase-in visitations in the same trade zone. This allows ships to avoid sailing back and forth between ports when transferring directly between the phase-out and phase-in.

6.1.2 Phase-in

We create visitations for each port call along a phase-in slot, and connect subsequent phase-in visitations to each other. The final visitation in a slot, which represents the time at which regular operations must begin on a service, is connected to the graph sink, τ.

The phase-in graph structure ensures that the goal service has a vessel in each of its slots. An example phase-in graph structure is portrayed in Figure 6.1, which shows an example graph for a service with three slots. Each sequence of visitations (colored red, green and blue) represents a slot on the goal service. Each visitation is labeled with the port and week that it is visited. The last node in each sequence corresponds to the on-time requirement (node $(c, 2)$) extended to each slot. After each of these visitations, the service begins normal operations, and is no longer under the control of the repositioning coordinator. This graph structure ensures that all vessels perform a legal phase-in, namely that each slot is assigned a single vessel. Each phase-in slot is guaranteed to be assigned a single vessel since there are as many slots as there are

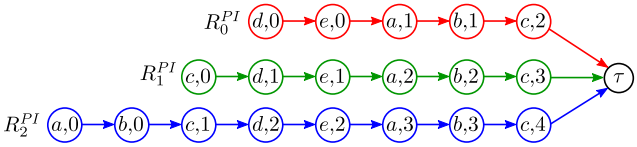


Figure 6.1: The phase-in graph structure for a service with 3 slots. Nodes are labeled (port, week). The sets R_0^{PI}, R_1^{PI}, and R_2^{PI} contain the nodes for each phase-in slot. Each set of nodes ends with a single visitation at port c on weeks 2, 3 and 4, ensuring that the weekly structure of the service is enforced.

vessels (three), the graph sink τ only has a single incoming node from each slot, and the paths of vessels are node disjoint (except for τ).

6.1.3 Flexible visitations

Flexible visitations are modeled by replicating each flexible visitation for each ship in the model. Flexible visitations are connected to all inflexible and flexible visitations within the same trade zone. Replicating flexible visitations for each vessel avoids requiring special constraints in the MIP model to handle the fact that multiple vessels can visit the same flexible visitation. This is because when a vessel visits a flexible visitation, the visitation must be assigned a time when it can take place. Simply copying the flexible nodes ensures that each flexible node can be scheduled along the path of a vessel with simple constraints. Since our instances generally do not contain many flexible visitations, this duplication does not significantly hinder the solvability of the instances. Note that this opens the possibility that two vessels may visit the same flexible visitation at the same time. We do not consider this to be a problem since flexible visitations are at ports that will probably have the capacity to deal with multiple ships. Since flexible visitations do not have fixed entry and exit times, the time required for a vessel to visit them must be taken into account. The total port stay at a flexible visitation consists of the *piloting time*, which is the time required to maneuver the vessel in to, and out of, a port, and the cargo/equipment (un)loading time.

6.1.4 Sail-on-service

We introduce a number of disjoint sets of graph arcs and graph nodes to represent a special graph structure that models SOS opportunities. We view an SOS as having three types of ports; *entry ports*, where vessels may join the SOS, *through ports*, in which a vessel must already be on the SOS, and *end ports* where a vessel may leave the SOS. The designations of these ports is left to the repositioning coordinator. We make this distinction in case there are circumstances outside the scope of the model that require certain ports to be called if an SOS is used.

Figure 6.2 shows the graph structure of an example SOS opportunity, o. Vessels may enter the SOS using arcs in the set $\hat{A}_o^{In}$, either through parallel sailing nodes (O_o^P) or transshipment nodes (O_o^{TS}), shown in red and green, respectively. Parallel

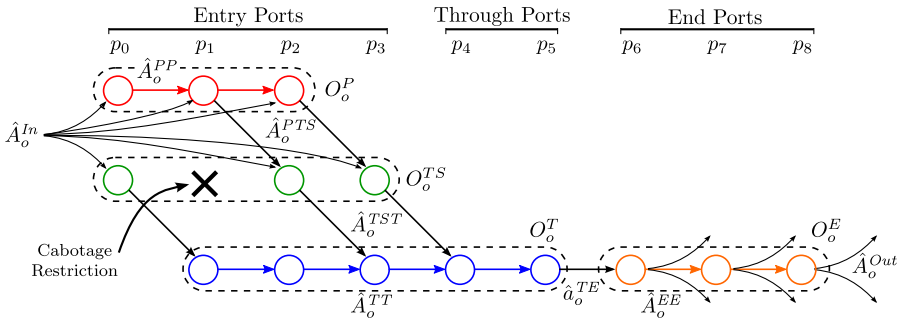


Figure 6.2: The graph structure of an example SOS opportunity, which contains parallel nodes O_o^P in red, transshipment nodes O_o^{TS} in green, a cabotage restriction at port p_1, through nodes O_o^T in blue, and end nodes O_o^E in orange.

sailings end in a transshipment, in which cargo is moved to the repositioning vessel. The parallel sailing nodes are connected to the transshipment nodes with the set of arcs $\hat{A}_o^{PTS}$. The set of arcs $\hat{A}_o^{PP}$ contains arcs connecting subsequent ports for parallel sailings. These arcs have twice the sailing cost for each vessel as the other arcs in the SOS graph structure, which have the normal sailing cost between two ports for each vessel. Note that p_3 has no parallel sailing node because transshipment is not allowed in p_4, which is a through port. Ports with cabotage restrictions, such as p_1, do not receive transshipment nodes, as transshipping at that particular port would violate the law. Transshipment nodes connect to through nodes (blue) using the arcs $\hat{A}_o^{TST}$. Once at a through node, a vessel must sail onward to the next through node using the arc set $\hat{A}_o^{TT}$, until it reaches the arc $\hat{a}_o^{TE}$. This arc connects the through nodes to the end nodes, and represents a bottleneck that ensures only one vessel uses a particular SOS. Since the paths of the vessels are node disjoint, two vessels would have to visit the latest node in O_o^T and the earliest node in O_o^E, which is not allowed. Finally, vessels may exit the SOS through the end nodes (orange) in the set O_o^E, using an arc in the set $\hat{A}_o^{Out}$. End nodes are connected by the arcs in the set $\hat{A}_o^{EE}$.

6.1.5 Sailing Cost

The fuel consumption of a ship is a cubic function of the speed of the vessel. We precompute the optimal cost for each inflexible arc using a linearized bunker consumption function, and compute the costs of flexible arcs during optimization in our MIP model. All inflexible arcs in the model are assigned a sailing cost for each ship that is the optimal sailing cost given the total duration of the arc. If the duration of the arc is greater than the time required to sail on the arc at a ship's minimum speed, the cost calculated using the minimum speed and the ship simply waits for the remainder of the duration. This is a common practice for shipping lines in order to add buffer to their schedules, thus making the network more robust to disruptions.

6.1.6 Graph Formalization

We now provide a formalized description of the graph based on our previous description. The following parameters are used to define the graph.

S	Set of ships, indexed by s.
V^i, V^f	Set of inflexible and flexible visitations, respectively.
A^i, A^f	Set of inflexible and flexible arcs, respectively.
A'	The set of arcs $(i,j) \in A$, where $i,j \in V'$.
L	Set of phase-in slots, where $\|L\| = \|S\|$, indexed by ℓ.
SOS	The set of SOS slots.
R_ℓ^{PI}	Set of visitations of phase-in slot $\ell \in L$.
R_s^{PO}	Set of phase-out visitations of vessel $s \in S$.
$O_o^{\{P,TS,T,E\}}$	Sets of parallel, transshipment, transit, and end visitations, with $o \in SOS$.
V^R	Set of non-SOS inflexible visitations, $V^R = \bigcup_{\ell \in l} R_\ell^{pi} \bigcup_{s \in s} R_s^{po}$.
TZ	Set of trade zones.
$z_i \in TZ$	Trade zone of visitation $i \in V$.
$t_i^E \in \mathbb{R}^+$	Time a vessel begins inflexible visitation $i \in V^i$.
$t_i^X \in \mathbb{R}^+$	Time a vessel ends inflexible visitation $i \in V^i$.
$\tau \in V$	Graph sink, which is not an actual visitation.
$V' = V \setminus \tau$	Set of nodes without the graph sink.
d_{ij}^{min*}	Minimum time required for any ship to sail from visitation i to j.
$A^{SD}(R)$	Set of arcs connecting subsequent visitations in the visitation set R.
A^{PO}	Set of arcs connecting phase-out slots to phase-in slots.
A^{PI}	Set of arcs from phase-in visitations to same trade zone phase-out visitations.
A^τ	Set of arcs from the final phase-in visitation to the graph sink.
$\hat{A}_o^{In}$	Set of arcs connecting to the start nodes of $o \in SOS$.
$\hat{A}_o^{Out}$	Set of arcs extending from the end nodes of $o \in SOS$.
$\hat{A}_o^{PTS}$	Set of arcs connecting the parallel nodes to transshipment nodes of $o \in SOS$.
$\hat{A}_o^{TST}$	Set of arcs connecting transshipment nodes to transit nodes of $o \in SOS$.
$\hat{A}_o^{TT}$	Set of arcs between transit nodes of $o \in SOS$.
$\hat{A}_o^{EE}$	Set of arcs between sequential end nodes of $o \in SOS$.
$\hat{a}_o^{TE}$	Arc from the latest transit node in $o \in SOS$ to its earliest end node.

We define the set of inflexible nodes as $V^i = \bigcup_{\ell \in l} R_\ell^{PI} \bigcup_{s \in s} R_s^{PO} \bigcup_{o \in SOS} (O_o^P \cup O_o^T \cup O_o^{TS} \cup O_o^E)$. The set of flexible visitations, V^f, contains all visitations that have equipment surpluses/deficits such that $V^f \cap V^i = \emptyset$. In order to formally define the set of arcs contained in the graph, let $follows(i,j) \in \mathbb{B}$ return *true* if and only if visitation j is scheduled on any service to immediately follow visitation i, with $i,j \in V^i$. In addition,

we let *can-sail*$(i,j) \in \mathbb{B}$ be *true* if and only if $t_j^E \geq t_i^X + \Delta_{ij}^{min*}$, where $i,j \in V'$. This indicates whether or not it is possible to sail between two visitations at the fastest speed of the fastest vessel in the model. Note that all of the arc sets are disjoint. We now formally define all of the previously mentioned sets of arcs.

$$
\begin{aligned}
A^{SD}(R) &= \{(i,j) \mid i,j \in R \wedge \mathit{follows}(i,j)\}, R \in \bigcup_{s \in S}\{R_s^{PO}\} \bigcup_{\ell \in L}\{R_\ell^{PI}\} \\
A^{PO} &= \{(i,j) \mid i \in \bigcup_{s \in S} R_s^{PO} \wedge j \in \bigcup_{\ell \in L} R_\ell^{PI} \wedge \mathit{can\text{-}sail}(i,j)\} \\
A^{PI} &= \{(i,j) \mid i \in \bigcup_{\ell \in L} R_\ell^{PI} \wedge j \in \bigcup_{s \in S} R_s^{PO} \wedge z_i = z_j \wedge \mathit{can\text{-}sail}(i,j)\} \\
A^{\tau} &= \{(i,\tau) \mid i \in \bigcup_{\ell \in L} \operatorname*{argmax}_{i' \in R_\ell^{PI}}\{t_{i'}^X\}\} \\
A^{f} &= \{(i,j) \mid \big((i \in V^f \vee j \in V^R) \wedge (i \in V^R \vee j \in V^f) \wedge (i \in V^f \vee j \in V^f)\big) \wedge z_i = z_j\} \\
\hat{A}_o^{In} &= \{(i,j) \mid i \in \bigcup_{s \in S} R_s^{PO} \wedge j \in (O_o^P \cup O_o^{TS}) \wedge \mathit{can\text{-}sail}(i,j)\} \\
&\qquad \bigcup \{(i,j) \mid i \in V^f \wedge j \in (O_o^P \cup O_o^{TS}) \wedge z_i = z_j \wedge \mathit{can\text{-}sail}(i,j)\} \\
\hat{A}_o^{Out} &= \{(i,j) \mid i \in O_o^E \wedge j \in \left(\bigcup_{\ell \in L} R_\ell^{PI} \bigcup_{o' \in \{SOS \setminus o\}} (O_{o'}^P \cup O_{o'}^{TS})\right) \wedge \mathit{can\text{-}sail}(i,j)\} \\
&\qquad \bigcup \{(i,j) \mid i \in O_o^E \wedge j \in V^f \wedge z_i = z_j \wedge \mathit{can\text{-}sail}(i,j)\} \\
\hat{A}_o^{PTS} &= \{(i,j) \mid i \in O_o^P \wedge j \in O_o^{TS} \wedge \mathit{follows}(i,j)\} \\
\hat{A}_o^{TST} &= \{(i,j) \mid i \in O_o^{TS} \wedge j \in O_o^{T} \wedge \mathit{follows}(i,j)\} \\
\hat{A}_o^{TT} &= \{(i,j) \mid i,j \in O_o^{T} \wedge \mathit{follows}(i,j)\} \\
\hat{A}_o^{EE} &= \{(i,j) \mid i,j \in O_o^{E} \wedge \mathit{follows}(i,j)\} \\
\hat{a}_o^{TE} &= (\operatorname*{argmax}_{i \in O_o^T}\{t_i^X\}, \operatorname*{argmin}_{j \in O_o^E}\{t_j^E\})
\end{aligned}
$$

The set of all arcs in the graph, A, is therefore defined by

$$
\begin{aligned}
A = &\bigcup_{s \in S}\left(A^{SD}(R_s^{PO})\right) \bigcup_{\ell \in L}\left(A^{SD}(R_\ell^{PI})\right) \cup A^{PI} \cup A^f \cup A^\tau \\
&\bigcup_{o \in SOS}\left(\hat{A}_o^{In} \cup \hat{A}_o^{Out} \cup A_o^{ST} \cup \hat{A}_o^{TT} \cup \hat{A}_o^{EE} \cup \hat{a}_o^{TE}\right).
\end{aligned}
$$

6.2 Arc Flow Model

We now define the MIP model that guides the vessels through the graph based on an arc flow approach, in which the amount of containers and equipment flowing on each

arc is explicitly modeled with the decision variables of the model. We use the following parameters and variables to supplement the parameters used to define the graph.

Parameters

S	Set of ships.
V'	Set of visitations minus the graph sink.
V^i, V^f	Set of inflexible and flexible visitations, respectively.
A^i, A^f	Set of inflexible and flexible arcs, respectively.
A'	Set of arcs minus those arcs connecting to the graph sink, i.e., $(i,j) \in A$, $i,j \in V'$.
Q	Set of equipment types. $Q = \{dc, rf\}$.
M	Set of demand triplets of the form (o,d,q), where $o \in V', d \subseteq V'$ and $q \in Q$ are the origin visitation, destination visitations and the cargo type, respectively.
$V^{q+} \subseteq V'$	Set of visitations with an equipment surplus of type q.
$V^{q-} \subseteq V'$	Set of visitations with an equipment deficit of type q.
$V^{q^*} \subseteq V'$	Set of visitations with an equipment surplus or deficit of type q ($V^{q^*} = V^{q^+} \cup V^{q^-}$).
$u_s^q \in \mathbb{R}^+$	Capacity of vessel s for cargo type $q \in Q$.
$M_i^{Orig}, (M_i^{Dest}) \subseteq M$	Set of demands with an origin (destination) visitation $i \in V$.
$v_s \in V'$	Starting visitation of ship $s \in S$.
$t_{si}^{Mv} \in \mathbb{R}$	Move time per TEU for vessel s at visitation $i \in V'$.
$t_i^E \in \mathbb{R}$	Enter time at inflexible visitation $i \in V'$.
$t_i^X \in \mathbb{R}$	Exit time at inflexible visitation $i \in V'$.
$t_i^P \in \mathbb{R}$	Pilot time at visitation $i \in V'$.
$r_q^{Var} \in \mathbb{R}^+$	Revenue for each TEU of equipment of type $q \in Q$ delivered.
$r^{(o,d,q)} \in \mathbb{R}^+$	Amount of revenue gained per TEU for the demand triplet.
$c_{sij}^{Sail} \in \mathbb{R}^+$	Fixed cost of vessel s utilizing arc $(i,j) \in A'$.
$c_{sij}^{VarSail} \in \mathbb{R}^+$	Variable hourly cost of vessel $s \in S$ utilizing arc $(i,j) \in A'$.
$c_i^{Mv} \in \mathbb{R}^+$	Cost of a TEU move at visitation $i \in V'$.
$c_{si}^{Port} \in \mathbb{R}$	Port fee associated with vessel s at visitation $i \in V'$.
$d_{ijs}^{Min} \in \mathbb{R}^+$	Minimum duration for vessel s to sail on flexible arc (i,j).
$d_{ijs}^{Max} \in \mathbb{R}^+$	Maximum duration for vessel s to sail on flexible arc (i,j).
$a^{(o,d,q)} \in \mathbb{R}^+$	Amount of demand available for the demand triplet.
$In(i) \subseteq V'$	Set of visitations with an arc connecting to visitation $i \in V$.
$Out(i) \subseteq V'$	Set of visitations receiving an arc from $i \in V$.
$\tau \in V$	Graph sink, which is not an actual visitation.

Variables

$w_{ij}^s \in \mathbb{R}_0^+$	The duration that vessel $s \in S$ sails on flexible arc $(i,j) \in A^f$.
$x_{ij}^{(o,d,q)} \in \mathbb{R}_0^+$	Amount of flow of demand triplet $(o,d,q) \in M$ on $(i,j) \in A'$.
$x_{ij}^q \in \mathbb{R}_0^+$	Amount of equipment of type $q \in Q$ flowing on $(i,j) \in A'$.
$y_{ij}^s \in \{0,1\}$	Indicates whether vessel s is sailing on arc $(i,j) \in A$.
$z_i^E \in \mathbb{R}_0^+$	Defines the enter time of a vessel at visitation i.
$z_i^X \in \mathbb{R}_0^+$	Defines the exit time of a vessel at visitation i.

Objective and Constraints

$$\max \; -\sum_{s\in S}\left(\sum_{(i,j)\in A'} c_{sij}^{Sail} y_{ij}^s + \sum_{(i,j)\in A^f} c_{sij}^{VarSail} w_{ij}^s\right) \tag{6.1}$$

$$+ \sum_{(o,d,q)\in M}\left(\sum_{j\in d}\sum_{i\in In(j)} \left(r^{(o,d,q)} - c_o^{Mv} - c_j^{Mv}\right) x_{ij}^{(o,d,q)}\right) \tag{6.2}$$

$$+ \sum_{q\in Q}\left(\sum_{i\in V^{q+}}\sum_{j\in Out(i)} \left(r_q^{Eqp} - c_i^{Mv}\right) x_{ij}^q - \sum_{i\in V^{q-}}\sum_{j\in In(i)} c_i^{Mv} x_{ji}^q\right) \tag{6.3}$$

$$- \sum_{j\in V'}\sum_{i\in In(j)}\sum_{s\in S} c_{sj}^{Port} y_{ij}^s \tag{6.4}$$

$$\text{s.t.} \sum_{s\in S}\sum_{i\in In(j)} y_{ij}^s \le 1 \qquad \forall j \in V' \tag{6.5}$$

$$\sum_{j\in Out(i)} y_{ij}^s = 1 \qquad \forall s \in S, i = v_s \tag{6.6}$$

$$\sum_{i\in In(\tau)}\sum_{s\in S} y_{i\tau}^s = |S| \tag{6.7}$$

$$\sum_{i\in In(j)} y_{ij}^s - \sum_{i\in Out(j)} y_{ji}^s = 0 \qquad \forall j \in \{V' \setminus \bigcup_{s\in S} v_s\}, s \in S \tag{6.8}$$

$$\sum_{(o,d,rf)\in M} x_{ij}^{(o,d,rf)} \le \sum_{s\in S} u_s^{rf} y_{ij}^s \qquad \forall (i,j) \in A' \tag{6.9}$$

$$\sum_{(o,d,q)\in M} x_{ij}^{(o,d,q)} + \sum_{q'\in Q} x_{ij}^{q'} \le \sum_{s\in S} u_s^{dc} y_{ij}^s \qquad \forall (i,j) \in A' \tag{6.10}$$

$$\sum_{i\in Out(o)} x_{oi}^{(o,d,q)} \le a^{(o,d,q)} \sum_{i\in Out(o)}\sum_{s\in S} y_{oi}^s \qquad \forall (o,d,q) \in M \tag{6.11}$$

$$\sum_{i\in In(j)} x_{ij}^{(o,d,q)} - \sum_{k\in Out(j)} x_{jk}^{(o,d,q)} = 0 \quad \forall (o,d,q) \in M, j \in V' \setminus (o \cup d) \tag{6.12}$$

$$\sum_{i \in In(j)} x^q_{ij} - \sum_{k \in Out(j)} x^q_{jk} = 0 \qquad \forall q \in Q, j \in V' \setminus V^{q^*} \tag{6.13}$$

$$d^{Min}_{ijs} y^s_{ij} \leq w^s_{ij} \leq d^{Max}_{ijs} y^s_{ij} \qquad \forall (i,j) \in A^f, s \in S \tag{6.14}$$

$$z^E_i = t^E_i \sum_{s \in S} \sum_{j \in In(i)} y^s_{ij} \qquad \forall i \in V^i \tag{6.15}$$

$$z^X_i = t^X_i \sum_{s \in S} \sum_{j \in Out(i)} y^s_{ij} \qquad \forall i \in V^i \tag{6.16}$$

$$z^X_i + \sum_{s \in S} w^s_{ij} \leq z^E_j \qquad \forall (i,j) \in A^f \tag{6.17}$$

$$\begin{aligned} &\sum_{(o,d,q) \in M^{Orig}_i} \sum_{j \in Out(o)} t^{Mv}_{so} x^{(o,d,q)}_{oj} + \sum_{(o,d,q) \in M^{Dest}_i} \sum_{d' \in d} \sum_{j \in In(d')} t^{Mv}_{sd} x^{(o,d,q)}_{jd'} \\ &+ \sum_{q \in Q} \left(\sum_{i' \in \{V^{q^+} \cap \{i\}\}} \sum_{j \in Out(i')} t^{Mv}_{si'} x^q_{i'j} + \sum_{i' \in \{V^{q^-} \cap \{i\}\}} \sum_{j \in In(i')} t^{Mv}_{sj} x^q_{ji'} \right) \\ &- z^X_i + z^E_i + t^P_i \sum_{j \in In(i)} y^s_{ij} \leq 0 \qquad \forall i \in V^f, s \in S \end{aligned} \tag{6.18}$$

The domains of the variables are as previously described. The objective consists of several components. The sailing cost (6.1) takes into account the precomputed sailing costs on arcs between inflexible visitations, as well as the variable cost for sailings to and from flexible visitations. Note that the fixed sailing cost on an arc does not only include fuel costs, but can also include canal fees or the time-charter bonus for entering an SOS. The profit from delivering cargo (6.2) is computed based on the revenue from delivering cargo minus the cost to load and unload the cargo from the vessel. Note that the model can choose how much of a demand to deliver, even choosing to deliver a fractional amount. We can allow this since each demand is an aggregation of cargo between two ports, meaning at most one container between two ports will be fractional. Equipment profit is taken into account in (6.3). Equipment is handled similar to cargo, except that equipment can flow from any port where it is in supply to any port where it is in demand. Finally, port fees are deducted in (6.4).

Multiple vessels are prevented from visiting the same visitation in (6.5). The flow of each vessel from its source node to the graph sink is handled by (6.6), (6.7) and (6.8), with (6.7) ensuring that all vessels arrive at the sink.

Arcs are assigned capacities if a vessel utilizes the arc in (6.9), which assigns the reefer container capacity, and in (6.10), which assigns the total container capacity, respectively. Note that constraints (6.9) do not take into account empty reefer equipment, since empty containers do not need to be turned on, and can therefore be placed anywhere on the vessel. Cargo is only allowed to flow on arcs with a vessel in (6.11). The flow of cargo from its source to its destination, through intermediate nodes, is handled by (6.12). Constraints (6.13) balance the flow of equipment in to and out of nodes. In

contrast to the way cargo is handled, equipment can flow from any port where it is in supply to any port where it is in demand. Since the amount of equipment carried is limited only by the capacity of the vessel, no flow source/sink constraints are required.

Flexible arcs have a duration constrained by the minimum and maximum sailing time of the vessel on the arc in (6.14). The enter and exit time of a vessel at inflexible ports is handled by (6.15) and (6.16), and we note that in practice these constraints are only necessary if one of the outgoing arcs from an inflexible visitation ends at a flexible visitation. Constraints (6.17) sets the enter time of a visitation to be the duration of a vessel on a flexible arc plus the exit time of the vessel at the start of the arc. Constraints (6.18) controls the amount of time a vessel spends at a flexible visitation. The first part of the constraint computes the time required to load and unload cargo and equipment, with the final term of the constraint adding the piloting time to the duration only if one of the incoming arcs is enabled (i.e., the flexible visitation is being used).

The model forms a disjoint path problem in which a fractional multicommodity flow is allowed to flow over arcs in the vessel paths, along with a small scheduling component in the flexible nodes. Flexible arcs could be alternatively represented using a discretized approach, however we forego a discretization because of the vast differences in timescales between port activities and sailing activities, which are on the order of hours and days, respectively. In order to achieve such a fine grained view of flexible arc activities, we would require numerous extra arcs and nodes for each flexible node.

6.3 Path-Based Model

The number of variables in the arc flow model grows based on the number of demands multiplied by the number of arcs. We formulate a path-based model that has even more variables, but can be solved in a structured manner using column generation that allows us to only generate those variables that are necessary. This means that the resulting problem is able to be solved with less variables than the arc flow approach. We use a Dantzig-Wolfe decomposition to split the problem into a master and subproblem. This decomposition only solves an LP, but the LSFRP is modeled by a MIP, meaning the path-based model is only able to find bounds on the optimal solution. However, on nearly all of the instances in our dataset we find integer solutions to the LP, meaning we find the optimal solution to most LSFRP instances with the path-based model.

We note that in order to use the following method, a starting solution is required. While an artificial solution can be used and then penalized, we generate a solution using the methods that we present later in Section 6.5.4.

6.3.1 Master Problem

We formulate our master problem as a set packing problem with a convexity constraint over the path chosen for each ship. We use the following parameters, variables, objective and constraints in the master and subproblem in addition to the parameters in model in Section 6.2.

Parameters

P^s	Set of all possible paths for ship s through the graph.
P	Set of all possible ship paths, i.e., $P = \bigcup_{s \in S} P^s$.
$\rho_p \in \mathbb{R}$	Reduced price of sailing, which is the profit of using path $p \in P$.
$\alpha_{ip} \in \{0, 1\}$	Indicates whether path $p \in P$ uses node $i \in V'$.
$\beta_{ijp} \in \mathbb{R}$	Sailing duration on flexible arc $(i, j) \in A^f$ on path $p \in P$.
$\gamma_{pd'}^{(o,d,q)} \in \mathbb{R}$	Amount of demand revenue for demand (o, d, q) delivered at $d' \in d$ on path $p \in P$.
$\delta_{ijp} \in \{0, 1\}$	Indicates whether path $p \in P$ uses arc $(i, j) \in A'$
$\eta_{ijp}^q \in \mathbb{R}$	Amount of equipment revenue for equipment type $q \in Q$ carried from $i \in V'$ to $j \in V'$ on path $p \in P$.

Variables

The model contains the variables $\lambda_p \in \{0, 1\}$, which indicate whether ship path $p \in P$ should be chosen by the model.

Objective and Constraints

$$\max \sum_{s \in S} \sum_{p \in P^s} \rho_p \lambda_p \tag{6.19}$$

$$\text{s. t.} \sum_{s \in S} \sum_{p \in P^s} \alpha_{ip} \lambda_p \leq 1 \qquad \forall i \in V' \tag{6.20}$$

$$\sum_{p \in P^s} \lambda_p = 1 \qquad \forall s \in S \tag{6.21}$$

The objective, (6.19), consists of the profit earned on a particular vessel path. Constraints (6.20) enforce the node disjoint structure of the graph. Together, (6.19) and (6.20) form the set packing component of the master problem. Each ship is assigned exactly one path in constraints (6.21), which represent the convexity constraint over the path variables for each ship.

6.3.2 Sub-problem

In order to compute the subproblem objective, we associate the dual variables π_i with constraints (6.20) and ψ_s with constraints (6.21). The profit of a path p for a ship s is given by

$$\hat{\rho}_p = \rho_p - \sum_{i \in V'} \alpha_{ip} \pi_i - \psi_s. \tag{6.22}$$

The constant ρ_p is the cost of a path in the arc-formulated problem. Let

$$\begin{aligned}\rho_p = &- \sum_{(i,j)\in A'} \delta_{ijp} c^{Sail}_{sij} - \sum_{(i,j)\in A^f} \beta_{ijp} c^{VarSail}_{sij} - \sum_{i\in V'} \alpha_{ip} c^{Port}_{si} \\ &+ \sum_{(o,d,q)\in M} \sum_{d'\in d} \gamma^{(o,d,q)}_{pd'} (r^{(o,d,q)} - c^{Mv}_{o} - c^{Mv}_{d'}) \\ &+ \sum_{q\in Q} \sum_{i\in V^{q+}} \sum_{j\in V^{q-}} \eta^{q}_{ijp} (r^{Eqp}_{q} - c^{Mv}_{i} - c^{Mv}_{j}) \end{aligned} \tag{6.23}$$

for some ship $s \in S$ and a path $p \in P^S$. We generate columns as long as $\exists p \in P$ such that $\hat{\rho}_p > 0$, which corresponds to finding the maximum profit path through the subproblem graph.

6.3.3 Reduced Graph

Although the subproblem is, in the worst case, just as hard as the original problem, we can reduce the size of the graph (and thereby the number of demands present). On some instances, up to 94% of the nodes and 96% of the arcs can be pruned for certain vessels, although for more instances between one fourth and two thirds of the nodes and arcs are pruned for each vessel. This generally makes the subproblem easier to solve. The graph size reduction is based on the fact that with only a single vessel present, the phase-out visitations of other vessels must no longer be a part of the graph, along with any visitations that are only reachable from those phase-outs. Additionally, several phase-in slots, SOS opportunities and visitations with equipment may not be usable due to the time when the vessel phases-out.

We formalize the graph size reduction as follows, using the same parameters as from the graph in Section 6.1 and arc flow model in Section 6.2. We are given a ship $s \in S$ for which to compute the reduced graph. Additionally, let $G^T = (V^T, A^T)$ be the transitive closure of the original graph, $G = (V, A)$. The reduced graph, $G^R = (V^R, A^R)$ with corresponding demands M^R is thus defined by:

$$V^R = \{i \in V' \mid (v_s, i) \in A^T\} \tag{6.24}$$

$$A^R = \{(i,j) \in A \mid i,j \in V^R\} \tag{6.25}$$

$$M^R = \{(o,d,q) \in M \mid o \in V^R \wedge \exists d' \in d, \text{ s.t. } d' \in V^R\} \tag{6.26}$$

The nodes of the reduced graph, defined by V^R in (6.24), are only those nodes which are reachable from the start visitation of the vessel (v_s). We then build the arc set A^R in (6.25) out of all arcs that have their start and end visitation in V^R. Finally, M^R in (6.26) stores all demands that have their origin and at least one destination in the reduced graph.

6.4 LSFRP with Inflexible Visitations

Many of the LSFRP instances we will later introduce do not have any flexible components, which means that a method that can exploit the fixed times of visitations can provide significant speed-ups versus a method that must also handle flexible visitations and arcs. We call this version of the problem the inflexible visitation LSFRP (IVLSFRP). In order to solve the IVLSFRP, we provide a *node flow* based model. This model exploits the fact that when all visitations are inflexible, a subset of visitations can only be sequenced in one way. We use this fact in order to model demands based on the graph nodes they can pass through. The following description is based on [150].

We provide two different node flow based models of the IVLSFRP, each of which uses a different modeling of equipment. In our first model, which we call the *equipment as flows* model, we do not change the way equipment is modeled from the arc flow model. We also provide a model in which we model equipment flows as demands, which we call the *equipment as demands* model.

In order to prevent a vessel from carrying too many containers, the amount of containers loaded on the vessel must be accounted for throughout its repositioning. In the arc flow model, this is done by explicitly keeping track of the amount of demand flowing on each arc. In contrast, the node flow model is able to keep count of the number of containers on the vessel implicitly based on the visitations on a vessel's path, and therefore only needs to determine how many containers from a particular demand are flowing. That is, instead of a variable for each demand on each arc, the node flow model has a variable for each demand on each vessel. In order to ensure a vessel is not overloaded over the course of its repositioning, it suffices to ensure it is not overloaded as it enters each visitation on its path, which corresponds to constraining the incoming arcs of a visitation. Since demands must be loaded and unloaded at visitations, which in the IVLSFRP have a fixed begin and end time, the times a demand can be loaded and unloaded are fixed as well. We can therefore determine which demands can be carried on each arc with a reachability analysis.

Since we know in advance which demands can be on the vessel at what times and places, we can post constraints for each visitation to ensure the vessel is not overloaded without explicitly modeling the path of the vessel. These constraints represent the state of a vessel as it enters a visitation, and they neither over or under constrain the problem. They are clearly sufficient to prevent the vessel from being loaded over capacity, since they cover all demands that can possibly be on a vessel as it enters each visitation on its path. They do not over constrain the problem because only those demands which can be loaded on the vessel are constrained. Due to the fixed visitations times, there is never a situation in which two demands are loaded in sequence on one potential vessel path, and are loaded simultaneously on a different path. This means an optimal solution can always be found, if the problem is not infeasible. Consider the following example.

Example 6.1. Figure 6.3 shows two graphs. In the first graph (a), node b has no fixed visitation time and must be scheduled, and in the second graph (b), all nodes have a

(a) A non-fixed time graph. (b) A fixed time graph.

Figure 6.3: Subsets of an LSFRP graph with potential vessel paths (red, blue).

fixed visitation time. A single vessel must sail through the graph, starting at a and ending at d. The demands in the instance are $\{(a, c), (b, d)\}$[2], meaning there is a demand from a to c and from b to d. First, consider the non-fixed time case in Figure 6.3a. The red path (a, b, c, d) and blue path (a, c, b, d) show two potential voyages of a vessel. On the red path, demand (a, c) is loaded while the vessel is at a, and then the vessel continues to b, where it loads (b, d). Thus, on the red path both demands are loaded on the vessel simultaneously. On the blue path, the vessel first loads the (a, c) demand, sails to c where it is delivered, and continues to b where it loads (b, d). In this case, the demands are loaded sequentially. Now consider the fixed time case in Figure 6.3b, in which the time of visiting b is known in advance. The red path visits nodes (a, b, c, d), and demand (a, c) and (b, d) are on the ship simultaneously at b. In fact, there is no path in the fixed time case where (a, c) and (b, d) can be loaded sequentially; they are either both on the ship or only (a, c) is on the ship.

In the LSFRP, a single demand can be delivered to any visitation in a set of destinations. These destinations all correspond to a single real-world port being visited by different services at different times. The arc flow model takes these multiple destinations into account by simply having each destination visitation of a demand act as a sink for that demand. This is advantageous for the model, since modeling each origin-destination pair individually would require many variables. However, in the node flow model, multiple destinations can cause a situation in which certain incoming arcs of nodes are over-constrained. This could result in the optimal solution not being found.

Example 6.2. Consider Figure 6.4, which shows a graph with the vessel path (a, b, e, f) shown with red, and the demands $\{(a, \{b, f\}), (e, \{f\})\}$, meaning there is a demand from a that may be delivered to either b or f, and a demand from e to deliver to f. Since it is possible for both demands to be flowing on arc (e, f), a constraint must be posted ensuring that $x^{(a,\{b,f\})} + x^{(e,\{f\})} \leq u_s^{dc}$. When a vessel travels on the path shown, the first demand, rather than being delivered to f, is dropped off at b. The vessel then continues to e where it loads the second demand. Consider the case where the capacity of the vessel is 50 TEU and both demands have 50 TEU available. Due to the constraint we must post on arc (e, f), we can only take a maximum of 50 TEU of both demands, even though it is possible to carry both demands in full.

[2]We ignore container types, as they are not relevant.

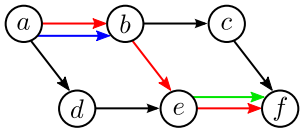


Figure 6.4: Sequential demand delivery in a multiple demand destination problem.

We remedy this problem by splitting each multiple destination demand into a demand for each origin-destination pair, and add a constraint in the node flow model to ensure that the amount of containers delivered to all of the destinations is not greater than the amount of demand available. In the following models, we use the set M' to represent the set of single origin-destination pairs of demands in the following node flow models. Formally, let $M' = \bigcup_{(o,d,q)\in M} \bigcup_{d'\in d}(o, d', q)$.

6.4.1 Preprocessing

We base our node based model on the same graph and notation as in Section 6.2. In order to model the flows based on nodes, we must determine which nodes a particular demand can traverse. We do this using a simple reachability analysis based on the transitive closure of the graph. Let $G^T = (V^T, A^T)$ be the transitive closure of the graph $G = (V, A)$, and

$$M_i^{Vis'} = \{(o, d, q) \in M' \mid (o, i) \in A^T \wedge \exists d' \in d \text{ s.t. } \big((i, d') \in A^T \vee i = d'\big)\}.$$

Each node is thereby assigned a set of demands ($M_i^{Vis'}$) based on whether the node is reachable from the demand origin and could visit at least one of the destinations of the demand. We further define $M_{i,rf}^{Vis'} = \{(o, d, q) \in M_i^{Vis'} \mid q = rf\}$ to be the set of reefer demands at node $i \in V'$. Using these sets of demands, we can now model demand flows on the nodes of the IVLSFRP graph.

6.4.2 Equipment as Flows

We extend the parameters in Section 6.2 with the following three parameters: M', the set of single origin single destination demands, $M_i^{Vis'}$, which is the set of demands (dry and reefer) that could traverse node i, and $M_{i,rf}^{Vis'}$, the set of reefer demands that could traverse node i.

Variables

$x_{ij}^t \in [0, \max_{s\in S} u_s^{dc}]$	Amount of equipment of type $q \in Q$ flowing on $(i, j) \in A'$.
$x_s^{(o,d,q)} \in [0, a^{(o,d,q)}]$	Amount of demand triplet $(o, d, q) \in M'$ carried on ship $s \in S$.
$y_{ij}^s \in \{0, 1\}$	Indicates whether vessel s is sailing on arc $(i, j) \in A$.

Objective and Constraints

$$\max \sum_{s \in S} \sum_{(o,d,q) \in M'} \left(r^{(o,d,q)} - c_o^{Mv} - c_d^{Mv} \right) x_s^{(o,d,q)} - (6.1) - (6.4) + (6.3) \tag{6.27}$$

s. t. (6.5); (6.6); (6.7); (6.8); (6.13);

$$x_s^{(o,d,q)} \leq a^{(o,d,q)} \sum_{i \in Out(o)} y_{oi}^s \qquad \forall (o,d,q) \in M', s \in S \tag{6.28}$$

$$x_s^{(o,d,q)} \leq a^{(o,d,q)} \sum_{d' \in d} \sum_{i \in In(d')} y_{id'}^s \qquad \forall (o,d,q) \in M', s \in S \tag{6.29}$$

$$\sum_{(o,d,q) \in M_i^{Vis'}} x_s^{(o,d,q)} + \sum_{q \in Q} \sum_{j \in In(i)} x_{ij}^q \leq u_s^{dc} \qquad \forall s \in S, i \in V' \tag{6.30}$$

$$\sum_{(o,d,q) \in M_{i,rf}^{Vis'}} x_s^{(o,d,q)} \leq u_s^{rf} \qquad \forall s \in S, i \in V' \tag{6.31}$$

$$\sum_{q \in Q} x_{ij}^q \leq \sum_{s \in S} u_s^{dc} y_{ij}^s \qquad \forall (i,j) \in A' \tag{6.32}$$

$$\sum_{d \in d'} x_s^{(o,d',q)} \leq a^{(o,d,q)} \qquad \forall s \in S, (o,d,q) \in M'. \tag{6.33}$$

The objective (6.27) contains the same calculation of sailing costs, equipment profits, and port fees as in the arc flow model. However, the demand profit is now computed using the demand flow variables. Note that unlike in M, all $(o,d,q) \in M'$ have $|d| = 1$. Thus, the constant c_d^{Mv} refers to the cost at a particular visitation.

We maintain constraints (6.5) through (6.8) and (6.13) directly from the arc flow model in order to enforce node disjointness along vessel paths and create the vessel and equipment flows. We refer readers to Section 6.2 for a full description of these constraints.

Constraints (6.28) and (6.29) allow a demand to be carried only if a particular vessel visits both the origin and a destination of the demand. Note that we do not need to limit the demands to be taken only by a single vessel because of the node disjointness enforced by constraints (6.5). Only a single vessel can enter the origin node of a demand, ensuring that only one vessel can carry a demand.

In constraints (6.30) and (6.31) we ensure that the capacity of the vessel is not exceeded at any node in the graph in terms of all containers and reefer containers, respectively. Equipment flows are handled in the dry capacity constraints (6.30). Due to the equipment balance constraints, ensuring that the equipment capacity is not exceeded at a node is sufficient for ensuring that the vessel is not overloaded.

Constraints (6.32) prevent equipment from flowing on any arc that does not have a ship sailing on it. When a vessel utilizes an arc, the constraint allows as much equipment to flow as the capacity of the vessel. When a vessel is assigned to an arc, the corresponding equipment flow variables have an upper bound of 0. And, finally, constraints (6.33)

ensure that the amount of demand carried for each single origin-destination demand does not exceed the amount of containers that are actually available.

6.4.3 Equipment as Demands

As an alternative to modeling equipment as flows, we present a model that creates demands for each equipment pair and adds them to M'. We let $M_t^{E'} = \{(o, \{d\}, dc) \mid o \in V^{q+} \wedge d \in V^{q-}\}$ be the set of demands corresponding to every pair of visitations with an equipment surplus/deficit for type $q \in Q$. Note that we set the demand type of all of the equipment demands to be dry (dc). We then append the equipment demands to M' as follows: $M' \leftarrow M' \cup \bigcup_{q \in Q} M_q^{E'}$. In addition, let $a^{(o,d,dc)} = \sum_{s \in S} u_s^{dc}$ and $r^{(o,d,dc)} = r_{dc}^{Eqp}$ for all $q \in Q, (o, d, dc) \in M_q^{E'}$. Thus, the maximum amount of equipment available for each equipment demand is equal to the sum of the capacities of all ships, and the revenue per TEU delivered is equal to the equipment revenue for each equipment type. Our model uses the same parameters as the arc flow model in Section 6.2 and the equipment as flow model in Section 6.4.2. The majority of the model is the same as the previous two models, however we include all of the objectives and constraints for completeness.

Objective and Constraints

We use the variables y_{ij}^s and $x_s^{(o,d,q)}$ from the equipment as flows model and require no additional variables. The model is as follows:

$$\max -(6.1) - (6.4) + (6.27) \tag{6.34}$$

subject to constraints (6.5), (6.6), (6.7), (6.8), from the arc flow model in Section 6.2, and (6.28), (6.29), (6.31), and (6.33) from the model in Section 6.4.2. Instead of the dry capacity constraint in the equipment as flows model, we impose the following constraint:

$$\sum_{(o,d,q) \in M_i^{Vis'}} x_s^{(o,d,q)} \leq u_s^{dc} \qquad \forall s \in S, i \in V'. \tag{6.35}$$

The objective, (6.34), combines the sailing costs and port fees from the arc flow model with the cargo demand objective from the equipment as demands model. Note the lack of any objective relating to equipment, as it is now a part of the demand structure.

As in the equipment as flows model, we include several constraints from the arc flow model to enforce node disjointness along vessel paths and control the vessel flows. However, we omit the equipment flow balance constraints (6.13). We also include the node demand constraints from the equipment as flows model, along with the reefer capacity constraint, as they are unaffected by modeling equipment as demands. We modify the dry capacity constraints (6.30) to produce constraints (6.35), in which the sum of the demands carried at a particular node must respect the vessel capacity.

6.5 Heuristic Approaches

Although solving the LSFRP to optimality is desirable given the large amounts of money at stake in the LSFRP, all of our optimal methods are only effective for problems up to a certain number of arcs and demands, as one would expect when dealing with an NP-complete problem. We therefore introduce two heuristic search procedures that use common initial solution generation approaches and local search neighborhoods. The two search procedures we use are simulated annealing (SA) [85] and late acceptance hill climbing (LAHC) [19], due to their good performance on routing and scheduling problems.

6.5.1 Simulated Annealing

We created a heuristic solution procedure for the LSFRP with cargo flows using an SA algorithm. We choose SA over other heuristic approaches because of its successful use in a wide range of scheduling and routing problems [140]. We forego large neighborhood approaches, such as adaptive large neighborhood search [127], because the vessel paths in the LSFRP are small subsets of the overall graph, meaning small changes are sufficient for finding good solutions. Our SA uses a penalized objective in which certain types of infeasible solutions are accepted in order to avoid getting stuck within the search landscape.

We use a combined reheating and restart strategy similar to the one used in [142] to overcome local optima. Reheating involves increasing the temperature of the SA after convergence to a factor of the initial temperature. We combine reheating with full restarts, in that we allow only several reheats before we restart the SA from the initial solution. The idea behind such a restart is that several reheats could put the solution of the SA in a part of the search space that is far away from the global optimum, and continual reheating may not move it in the correct direction. Restarting from the initial solution allows the SA to move in a different direction and find a better solution. Our solution procedure can be viewed as a form of iterated local search [97], in which reheating is the perturbation procedure and the local search to be iterated over is simulated annealing.

Algorithm 6.1 shows the particular version of the SA algorithm that we are using in this work, parameterized as follows: p represents the problem to solve, f is the objective evaluation function, t^{Init} is the initial temperature, α is the temperature reduction factor, β is the reheating factor, t^{Min} is the convergence temperature, r^{Itrs} is the maximum number of non-improving iterations before reheating, $r^{Restart}$ is the number of reheats before resetting the solution to the initial solution, r^{Reheat} is the number of non-improving reheats before stopping the search.

After creating an initial solution on line 2 and initializing variables on the following lines, the algorithm begins its outer reheat/restart loop. On lines 8–10 we reset the solution used for the current reheat if the number of reheats exceeds a parameter $r^{Restart}$. This allows the SA to choose a different path from the starting solution, one that could

Algorithm 6.1 The SA algorithm with reheating and restarts.

```
1: function SA(p, f, t^Init, α, β, t^Min, r^Itrs, r^Restart, r^Reheat )
2:     s^Init ← CreateSolution(p)
3:     s* ← s^Init; s*_prev ← s^Init
4:     t ← t^Init
5:     reheats ← 0; nonImprovingReheats ← 0
6:     repeat                                                  ▷ Reheat/restart loop.
7:         nonImprovingItr ← 0;
8:         if reheats ≥ r^Restart then
9:             s ← s^Init
10:            reheats ← 0
11:        repeat                                              ▷ SA loop.
12:            s' ← SelectNeighbor(s)
13:            if f(s') > f(s) then s ← s'
14:            else
15:                if exp((f(s')−f(s))/t) > Random() then s ← s'   ▷ Metropolis criterion.
16:                nonImprovingItr ← nonImprovingItr + 1
17:            if f(s') > f(s*) then s* ← s'
18:            t ← tα
19:        until t < t^Min or nonImprovingItr ≥ r^Itrs
20:        t ← t^Init β                  ▷ Reheat to a factor β of the initial temperature.
21:        reheats ← reheats + 1
22:        if s* ≤ s*_prev then nonImprovingReheats ← nonImprovingReheats + 1
23:        s*_prev ← max{s*, s*_prev}
24:    until Time limit reached or nonImprovingReheats ≥ r^Reheat
25: return s*
```

lead to better solutions. Within the SA inner loop on lines 11–19, a random neighbor is selected and replaces the current solution if it has a higher objective value. When the objective of the new solution is worse than that of the current solution, we accept the solution according to the Metropolis criterion [85] (line 15). We then update the incumbent solution (line 17) and reduce the temperature on an exponential cooling schedule according to the factor α on line 18. We exit the inner loop if the temperature falls below the threshold, t^{Min}, or the number of iterations in which no improving solution was found is greater than r^{Itrs}. We reheat the temperature to a factor β of the original temperature and continue the search until either running out of CPU time or we exceed the maximum number of non-improving reheats, r^{Reheat}.

6.5.2 Late Acceptance Hill Climbing

The LAHC algorithm was first introduced in [19] and was further investigated in [1, 20, 112, 160]. LAHC consists of a standard hill climbing (i.e., greedy local search) algorithm with one key difference: the acceptance criterion compares candidate solution to a solution ℓ iterations ago, in addition to the solution from the previous iteration.

Algorithm 6.2 The LAHC algorithm, based on the pseudocode in [20].

```
1: function LAHC(p, f, ℓ)
2:     iter ← 0
3:     s* ← CreateSolution(p)
4:     s ← s*
5:     sols ← Array(ℓ)
6:     for k ∈ {0, ..., ℓ − 1} do sols[k] ← f(s*)
7:     repeat
8:         v ← iter mod ℓ
9:         s′ ← SelectNeighbor(s)
10:        if f(s′) > f(sols[v]) or f(s′) > f(s) then s ← s′
11:        if f(s′) > f(s*) then s* ← s′
12:        sols[v] ← s
13:        iter ← iter + 1
14:    until Convergence criteria met
15: return s*
```

Thus, LAHC is allowed to accept degenerating solutions as long as their objective is less than the objective evaluation ℓ iterations ago, helping to prevent LAHC from becoming trapped in a local optimum as in the case of a standard hill climbing algorithm.

Algorithm 6.2 shows a generic LAHC procedure. The LAHC function accepts three parameters: the optimization problem to solve, p, the objective evaluation (i.e., the fitness function), f, and the number of solutions to store, ℓ. The LAHC is initialized in lines 2 – 7. First, an initial solution is constructed. Then, an array for storing the last ℓ solutions is initialized to store ℓ copies of the initial solution. The following procedure is then performed (lines 9 – 16) until some set of convergence criteria is satisfied. The index of the solution ℓ iterations ago is stored in v, and a candidate solution, s', is proposed by the SelectNeighbor(s) function. The candidate solution is accepted if the objective of s' is greater than the objective of the solution ℓ iterations ago, or the objective is greater than the immediately previous solution. The incumbent s^* is then updated to store the best solution seen so far. Finally, the solution is stored in the solution list and the iteration counter is incremented.

We use the generic LAHC procedure within the same restart procedure as the SA. Replacing lines 11 – 19 in Algorithm 6.1 with the LAHC procedure in Algorithm 6.2 gives the LAHC with restarts procedure that we use in this work (ignoring temperature considerations from the SA).

6.5.3 Solution Representation

Both SA and LAHC represent a solution to the LSFRP as a set of sequences of nodes from the graph, V'. Each sequence stores the nodes that a particular ship visits. Formally, a solution to the LSFRP is represented by $\bigcup_{s \in S}\{(v^s_1, \ldots, v^s_{\chi_s})\}$, where χ_s is the number of visitations on the path of ship s and $v^s_1, \ldots, v^s_{\chi_s} \in V'$.

We allow the following types of infeasible solutions in both the SA and LAHC approaches, and penalize such solutions in the objective. Any solution must enforce all of the constraints of the LSFRP (as described in Section 6.2) except for two groups of constraints. First, we soften the node disjointness constraints (6.5) and allow multiple vessels to visit the same visitation, subject to a penalty for each violation. Note that we still enforce node disjointness along a single vessel path, i.e., no loops are allowed. Second, we allow paths that represent temporally infeasible routes, which corresponds to softening constraints (6.15) – (6.18). Both of these cases allow the local search to operate on a single path at a time, in that it lets ships sail to visitations that are profitable without directly dealing with the feasibility repercussions of doing so.

6.5.4 Initial Solution Generation

We provide three solution generation heuristics that generate an initial solution from which the local search can start. Our heuristics provide a diversity of approaches, ranging from generating simple paths as in the direct route heuristic, to constructing solutions while taking into account cargo flows in the greedy and shortest path heuristics.

Direct route heuristic (DRH)

We model the connections between the starting visitation of each vessel with all of the feasible phase-ins in a linear assignment problem. The cost of each vessel/phase-in assignment is equal to the sailing cost of the vessel to the particular phase-in if the sailing is feasible, or infinity if it is not. We solve the following problem with the following parameters and variables using notation from our graph in Section 6.1. Let τ_ℓ be the latest visitation in slot $\ell \in L$. That is, $\tau_\ell = \text{argmax}_{i \in R_\ell^{PI}}\{t_i^E\}$. We also let

$$cost(s,i,j) = \begin{cases} c_{sij}^{Sail} & \text{if } (i,j) \in A' \\ \infty & \text{otherwise} \end{cases}$$

be the fixed cost of sailing ship $s \in S$ from $i \in V'$ to $j \in V'$, if the sailing is feasible. If there is no sailing between i and j, the cost is set to infinity. The decision variables $x_{s\ell} \in \{0,1\}$ indicate whether ship $s \in S$ should utilize phase-in $\ell \in L$.

$$\min \sum_{s \in S} \sum_{\ell \in L} cost(s, v_s, \tau_\ell) x_{s\ell} \tag{6.36}$$

$$s.t. \sum_{s \in S} x_{s\ell} = 1 \qquad \forall \ell \in L \tag{6.37}$$

$$\sum_{\ell \in L} x_{s\ell} = 1 \qquad \forall s \in S \tag{6.38}$$

$$x_{s\ell} \in \{0,1\} \tag{6.39}$$

The objective (6.36) minimizes the cost of the arcs from the phase-out to the phase-in chosen. Constraints (6.37) ensures that every vessel is assigned exactly one phase-in, and constraints (6.38) guarantee that every phase-in is assigned a ship.

The direct route heuristic generates a feasible starting solution, but it is rarely, if ever, an optimal solution to the LSFRP. Nonetheless, the solution provides a reasonable blank slate from which vessel routes can be expanded. We use the Hungarian algorithm [91] to solve the linear assignment problem.

Shortest paths heuristic (SPH)

We generate paths for vessels using a shortest path algorithm that iteratively creates a path for each ship on a graph containing only those visitations that are not visited in any previously generated path. While this ensures that any solution the heuristic generates will be node disjoint[3], it may not always be possible to generate any solution at all since all of the feasible phase-in opportunities for a particular ship may already have been assigned to another ship. We order the ships by their first possible phase-out time and generate paths starting with the ship with the latest phase-out. We found that with this ordering, only one out of our 44 instances could not generate a feasible starting solution, whereas with random orderings or with an ascending phase-out time ordering, feasible starting solutions could almost never be generated. If it is not possible to generate a solution, we fall back on DRH, although this never happened in our experiments.

Sailing costs on inflexible arcs as well as port fees are easy to take into account in the shortest path algorithm, however flexible arcs and cargo/equipment revenues pose a challenge. It is not possible to take these components fully into account within a standard shortest path algorithm, since this would require the cost of a particular arc to vary based on the scheduling of flexible arcs. Thus, the heuristic generates solutions that do not re-use visitations between vessels, but are not necessarily temporally feasible on instances with flexible arcs.

Flexible arc handling We allow flexible arcs to be used in the shortest path, even though they represent a scheduling problem that cannot be solved while the shortest path algorithm is running. We ignore temporal feasibility and focus only on the cost of the flexible arc. We define a parameter γ in the range $[0, 1]$ that represents how fast the ship is sailing over its minimum speed, and we assign arcs a sailing cost based on the speed of the vessel. This allows the heuristic to try to take into account some of the costs that would be incurred using a flexible arc.

Cargo handling The profit for each cargo demand in the graph is represented by computing the total possible profit from the demand (cargo revenue less move fees at the origin and destination) and multiplying this value by a scaling parameter ℓ^{Cargo},

[3]Note that the solution generated may be temporally infeasible.

which is in the range $[0, 1]$. We then offset the sailing costs and port fees at the origin and destination visitations using this scaled cargo profit. Thus, nodes where lots of cargo originates or is delivered have a high profit and are desirable for the shortest path algorithm to visit. Since cargo can only be delivered if both the origin and destination are on the path of a ship, this heuristic cannot guarantee that the path taken is actually one that has profitable cargo flows.

Equipment handling We perform a similar process of adding profit to visitations with equipment surpluses/deficits. The primary difference is that we do not know how much equipment can be loaded on to the vessel. Thus, we introduce a parameter $\ell^{Eqp} \in \mathbb{Z}^+$ that represents the amount of equipment to load or unload at a visitation. We cannot guarantee the ship will have sufficient capacity to actually load or unload that amount of cargo, but this allows the shortest path algorithm to visit visitations with equipment, which might otherwise be ignored.

Shortest Path Implementation Since we take into account cargo and equipment profits in this heuristic, arcs can have costs or revenues associated with them, meaning the sign of all the arcs is not the same. We therefore use the Bellman-Ford algorithm to sequentially compute the shortest path for each vessel. After a shortest path is computed, we update a list of banned visitations that may not be used again. Any visitation that is banned is considered to have a distance of infinity to and from all visitations, ensuring the shortest path algorithm does not choose it. It is possible that negative cycles are generated by the algorithm. We handle these by clearing the list of banned visitations and starting the algorithm from the first vessel in the vessel ordering using updated cargo and equipment profit parameters. After each failure, we subtract 0.1 from the cargo profit parameter ℓ^{Cargo} and 50 from ℓ^{Eqp}, until these parameters hit 0. When both parameters are 0, the graph is guaranteed to not have any negative cycles since arcs reflect only sailing costs. In practice, this is only necessary on several instances.

Greedy heuristic (GH)

The greedy heuristic (GH) chooses the most profitable outgoing arc from each visitation based on the same profit calculations and parameters as SPH. Similar to SPH, GH does not allow vessels to visit the same visitation more than once, and does this by storing a list of banned nodes after computing a greedy path for a vessel. The order the paths are generated in is the same as in SPH, as nearly every other ordering we tried resulted in failure of the algorithm to compute a solution. As in SPH, if a solution cannot be generated we fall back on DRH. We also tried creating a greedy heuristic that is not concerned with feasibility, however, we found that the solutions it generates tend to be of poor quality, as many vessels sail to the same phase-in visitation. This means the search procedure then wastes time trying to fix the phase-in infeasibility. This time can be better spent searching for a good solution.

6.5.5 Neighborhoods

At each iteration of the SA and LAHC methods, one of the following neighborhood operators is chosen uniformly at random to modify the current solution. We save more intelligent neighborhood selection schemes, such as the adaptive strategy used in adaptive large neighborhood search [127], for future work.

Visitation addition A ship, s, is selected uniformly at random along with an arc (u, v) on the path of s. A new visitation, w, is chosen such that an arc exists from u to w and from w to v, i.e., $w \in out(u)$ and $w \in in(v)$, and w is not already on the path of s. The visitation w is then inserted into the path of s between u and v. If no such visitation w exists, then the neighborhood performs no changes.

Visitation removal A ship, s, is selected uniformly at random along with a visitation on its path, u, such that u is neither the first or last visitation on the path. Visitation u is removed from the path if there exists an arc from the visitation before u to the visitation after u. If no such arc exists, the solution is not changed.

Visitation swap Two ships s and s', $s \neq s'$ are chosen uniformly at random, and a visitation u is selected from the path of s. If a visitation w on the path of s' exists such that swapping u and w is possible, i.e., $In(u) \cap In(w) \neq \emptyset$ and $Out(u) \cap Out(w) \neq \emptyset$, and swapping u and v would not introduce a duplicated node on either path, then u and w are swapped between paths.

Random path completion (RPC) A ship, s, is selected uniformly at random along with a visitation, u, on its path. All visitations subsequent to u are removed from the path of s, and are replaced with a random path from u to the graph sink. Each visitation added to the random path must not already be on the path of s, to ensure there are no loops over flexible visitations. If it is impossible to finish a random path without containing a loop, the random path is abandoned and the solution is not changed.

Demand destination completion (DDC) A ship, s, is selected uniformly at random along with a visitation on its path, u, from which demand is loaded. A demand is chosen that could be loaded at u, but cannot be delivered because none of its delivery visitations are on the path of s. The neighborhood attempts to connect the current path to one of the destinations of the cargo using a breadth first search. Then, another breadth first search is started from the destination back to any subsequent visitation on the path, v. If no such path exists, or such paths can only be created by introducing a duplicated node into the vessel's path, then the solution is left unchanged. All nodes in between u and v are deleted from the path and replaced with the nodes from the breadth first searches.

6.5.6 Objective Evaluation

After applying a neighborhood operator to a solution and generating a candidate solution, the search procedure must update its objective value for the current solution. We exploit the fact that the paths of vessels are disjoint and only update the objective for the paths that a neighborhood operator changes. Note that since we admit infeasible solutions as described above, it is possible that some cargo is carried multiple times. However, the extra revenue that vessels can gain is significantly less than the penalty for multiple vessels calling the same visitation.

The objective in the LSFRP consists of three components: the easy to calculate fixed costs for inflexible arcs and port fees, the cost of sailing on flexible arcs, and the profit from delivering cargo and equipment. Computing the cost of using flexible arcs requires solving a simple scheduling problem for the vessel. Since the amount of time it takes to load cargo and equipment is taken into account at flexible visitations, the scheduling of flexible arcs and the handling of cargo and equipment must be solved together. We therefore formulate a linear program to compute the objective and load the most profitable cargo along the vessel's path, and solve it using CPLEX. We adopt the same notation as for the previous mathematical models.

We compute the cost of each ship's path independently. Given the path of ship s as per our solution representation, $\mathcal{V}_s = (v^s_1, \ldots, v^s_{\chi_s})$, the cost of the path is defined by an LP, and $\mathcal{V}^f_s = \mathcal{V}_s \cap V^f$ be the flexible nodes used on the path. We also let $\mathcal{A}_s = ((v^s_1, v^s_2), (v^s_2, v^s_3), \ldots (v^s_{(\chi_s-1)}, v^s_{\chi_s}))$ be the sequence of arcs used on the path of ship s, and $\mathcal{A}^f_s = \mathcal{A}_s \cap A^f$ be the flexible arcs used on the path. Since the ship's path is fully defined by $\mathcal{V}_s$, we can pre-compute which demands and equipment flows can be carried by ship s. We merge equipment flows into the demand structure as in the equipment as demands IVLSFRP model as this simplifies the constraints in the problem. Let

$$M^{LS}_s = \{(o,d,q) \in M \mid o \in \mathcal{V}_s \wedge \exists d' \in d \text{ s.t. } d' \in \mathcal{V}_s\} \bigcup_{q \in Q} E_q$$

be the set of demands and equipment that can be carried by ship s, where

$$E_q = \{(o,d,dc) \mid o \in V^{q+} \cap \mathcal{V}_s \wedge d \in V^{q-} \cap \mathcal{V}_s\}$$

is the set of origin-destination pairs for equipment that is available on the vessel path. For any $(o,d,dc) \in E_q, \forall q \in Q$, we let $r^{(o,d,dc)} = r^{Eqp}_q$ and $a^{(o,d,dc)} = u^{dc}_s$. This sets the revenue of delivering a demand representing equipment to the per TEU equipment delivery revenue, and the amount of equipment available to be the dry capacity of the ship, respectively.

In order to keep track of the portion of the objective penalizing infeasibility, we let $g^s_i = |\{s' \in S \setminus \{s\} \mid i \in \mathcal{V}_{s'}\}|$, which is the number of paths in which visitation i is contained, other than the path of ship s. The constant h_i is equal to 1 iff $i \in \bigcup_{\ell \in L} R^{PI}_\ell$, meaning i is a phase-in visitation.

Parameters Our LP uses the following parameters.

$\mathcal{V}_s, \mathcal{V}_s^f$	Sequence of visitations on the path of ship $s \in S$.
$\mathcal{A}_s, \mathcal{A}_s^f$	Sequence of arcs on the path of ship $s \in S$.
Q	Set of equipment types. $Q = \{dc, rf\}$.
M_s^{LS}	Set of demands and equipment that can be carried by ship $s \in S$, consisting of demand triplets (o, d, q) defined as in Section 6.2.
$c_{sij}^{Sail} \in \mathbb{R}^+$	Fixed cost of vessel s utilizing arc $(i, j) \in A'$.
$c_{sij}^{VarSail} \in \mathbb{R}^+$	Variable hourly cost of vessel $s \in S$ utilizing arc $(i, j) \in A'$.
$c_i^{Mv} \in \mathbb{R}^+$	Cost of a TEU move at visitation $i \in V'$.
$c_{si}^{Port} \in \mathbb{R}$	Port fee associated with vessel s at visitation $i \in V'$.
$r^{(o,d,q)} \in \mathbb{R}^+$	Amount of revenue gained per TEU delivered for the demand triplet (o, d, q).
$d_{ijs}^{Min}, d_{ijs}^{Max} \in \mathbb{R}^+$	Minimum and maximum duration for vessel s to sail on flexible arc (i, j), respectively.
$t_i^E, t_i^X \in \mathbb{R}$	Enter and exit time at inflexible visitation $i \in V'$, respectively.
$t_i^P \in \mathbb{R}$	Pilot time at visitation $i \in V'$.
$t_{si}^{Mv} \in \mathbb{R}$	Move time per TEU for vessel s at visitation $i \in V'$.
$a^{(o,d,q)} \in \mathbb{R}^+$	Amount of demand available for the demand triplet.
$u_s^q \in \mathbb{R}^+$	Capacity of vessel s for cargo type $q \in Q$.
g_i^s	Number of paths in which visitation i is included, not including the path of ship s.
h_i^s	Equal to 1 iff visitation i is a phase-in visitation.

Variables Our LP uses the following variables.

$z_i^E \in \mathbb{R}^+$	Enter time at visitation $i \in \mathcal{V}_s$
$z_i^X \in \mathbb{R}^+$	Exit time at visitation $i \in \mathcal{V}_s$
$x^{(o,d,q)} \in [0, a^{(o,d,q)}]$	Amount of demand (o, d, q) carried.

Objective and constraints The LP is defined as follows.

$$\max \sum_{(o,d,q) \in M_s^{LS}} (r^{(o,d,q)} - c_o^{Mv} - \max_{d' \in d}\{c_{d'}^{Mv}\}) x^{(o,d,q)} \tag{6.40}$$

$$- \sum_{(i,j) \in \mathcal{A}_s} c_{sij}^{Sail} - \sum_{(i,j) \in \mathcal{A}_s^f} c_{sij}^{VarSail}(z_j^X - z_i^E) - \sum_{i \in \mathcal{V}_s} c_{si}^{Port} \tag{6.41}$$

$$- \sum_{i \in \mathcal{V}_s} g_i^s \left((1 - h_i)p + h_i p^{PI}\right) \tag{6.42}$$

$$\text{subject to} \quad d_{ijs}^{Min} \leq z_j^X - z_i^E \leq d_{ijs}^{Max} \qquad \forall (i,j) \in \mathcal{A}_s^f \tag{6.43}$$

$$z_i^E \leq z_i^X \qquad \forall i \in \mathcal{V}_s \tag{6.44}$$

$$z_i^E = t_i^E \qquad \forall i \in \mathcal{V}_s \setminus \mathcal{V}_s^f \tag{6.45}$$

$$z_i^X = t_i^X \qquad \forall i \in \mathcal{V}_s \setminus \mathcal{V}_s^f \qquad (6.46)$$

$$z_i^X - z_i^E - t_i^P - \sum_{(i,d,q)\in M_s^{LS}} t_{si}^{Mv} x^{(i,d,q)} - \sum_{\{(o,d,q)\in M_s^{LS} \mid i\in d\}} t_{si}^{Mv} x^{(o,d,q)} \geq 0 \qquad \forall i \in \mathcal{V}_s^f \qquad (6.47)$$

$$\sum_{(o,d,q)\in M_i^{Vis\prime}} x^{(o,d,q)} \leq u_s^{dc} \qquad \forall i \in \mathcal{V}_s \qquad (6.48)$$

$$\sum_{(o,d,rf)\in M_{i,rf}^{Vis\prime}} x^{(o,d,rf)} \leq u_s^{rf} \qquad \forall i \in \mathcal{V}_s \qquad (6.49)$$

The objective is to maximize the container carrying profit (including equipment) in (6.40) minus the sailing costs, both flexible and inflexible, and port fees in (6.41), and minus the penalization of infeasibility in (6.42). We apply the penalization factor p for each path that visitation i is contained in other than that of ship s if the visitation is not a phase-in visitation. If the visitation is a phase-in visitation, we apply the penalty p^{PI}. Note that the fixed sailing cost and port fees components of objective (6.41) are simply constants that can be computed before computing the LP.

Constraints (6.43) require that the sailing time on flexible arcs is between the minimum and maximum sailing speed of the ship. Constraints (6.44) ensure that the entrance time of a ship at a visitation is earlier than its exit time. Constraints (6.45) and (6.46) fix the times of inflexible visitations on the path. Note that we replace these variables with constants before passing the model into CPLEX to improve the solution speed. The loading and unloading time of containers at flexible visitations is taken into account in constraints (6.47). Constraints (6.48) and (6.49) prevent the vessel from loading too many dry and reefer containers or reefer containers, respectively. The bounds of the variables are as defined.

If the model is temporally infeasible, we ignore cargo and equipment and penalize the objective by the constant p^T. We choose this over making the timing constraints sort because the sailing costs are not correctly computed in such a model, and even offers some incentive for infeasibility.

Greedy objective computation

In certain situations it is possible to avoid calling CPLEX and thereby speed up the computation of the objective. If a vessel's path includes no flexible visitations and at no visitation could the amount of cargo (both dry and reefer) loaded on to the vessel exceed its capacity, then we can use a simple greedy algorithm to load cargo onto the vessel. The greedy algorithm works by loading all available cargo at all visitations on the path, and then fills the remaining capacity of the vessel with equipment. This will always be the optimal loading of cargo and equipment as long as the profit earned per TEU from carrying equipment is less than the profit per TEU of any demand. In practice this is true, since customer's cargo is preferred over empty containers. We then combine the cargo profit computed by the greedy algorithm with the sailing costs and port fees, which do not require an LP to compute.

6.6 Computational Complexity

We reduce the knapsack problem to the LSFRP with flexible visitations in order to show that the LSFRP with flexible visitations is weakly NP-complete. Given n items, each with a profit p_i and a size s_i, and a knapsack with a capacity C, the knapsack problem maximizes the objective $\sum_{i=0}^{n} p_i x_i$ where x_i is a binary variable indicating whether or not item i is in the knapsack, subject to the capacity constraint $\sum_{i=0}^{n} s_i x_i \leq C$.

Theorem 6.1. *The LSFRP with flexible visitations is weakly NP-complete.*

We first note that the LSFRP is clearly in NP, as the total profit can be easily computed from the paths of the vessels through the graph. We initialize an LSFRP with a single vessel and no cargo or equipment demands. The problem instance contains a single phase-out visitation, ω, and a single phase-in visitation, λ. The port fees at both ω and λ are 0, and we let $t^E_\omega = t^X_\omega = 0$ and $t^E_\lambda = t^X_\lambda = C$. In other words, the timespan in which the repositioning must take place is limited to the capacity of the knapsack. For each knapsack item, we create a flexible visitation, f_i, which has a duration of exactly s_i, i.e., $t^P_{f_i} = s_i$. The port fee for visiting f_i is $-p_i$, since the LSFRP maximizes profit (i.e., minimizes fees). All flexible nodes, as well as ω and λ, are in a single trade zone. Therefore, the specification of the LSFRP graph ensures that the phase-out node, ω, connects to all flexible nodes, all flexible nodes connect to each other, and all flexible nodes connect to the phase-in node, λ. The sailing time of the vessel between all nodes in the graph is set to 0.

Item i is included in the knapsack solution if and only if the vessel visits f_i during its repositioning. Since the vessel can only visit a single flexible visitation at a time, the duration of each flexible visitation is fixed to the size of the item it represents, and the phase-in visitation is fixed in time to the size of the knapsack, the capacity constraint of the knapsack must be satisfied. Additionally, according to the objective of the LSFRP, only the flexible visitations corresponding to the maximum profit knapsack items will be chosen. Therefore, the LSFRP with flexible visitations is NP-complete. □

We note that flexible ports are not present in every LSFRP problem, and this proof only covers those with flexible ports. This is not to say that LSFRP problems without flexible ports are necessarily polynomial time solvable. Indeed, the LSFRP without flexible ports is likely NP-complete, however this is not trivial to prove and remains an open problem at this time.

6.7 Computational Evaluation

We evaluate the performance of our our various approaches to the LSFRP across two datasets of real-world and real-world inspired instances. One of the datasets contains confidential data from our industrial collaborator, thus we created a second dataset with an anonymized version of the confidential data. We are able to solve all but two instances on the confidential dataset, and all but two instances on the public dataset,

to optimality. Our SA algorithm finds optimal solutions on many instances in our dataset, and solutions with low optimality gaps on the rest of the instances. We also find high quality solutions on our industrial collaborator's reference scenario that earn an additional profit of $14 million over the reference solution. All of the computational results in this section were performed using AMD Opteron 2425 HE processors with CPLEX 12.4 and a maximum of 10GB of RAM.

6.7.1 Dataset

We created a benchmark set of instances containing two real world repositioning scenarios, with three and eleven ships, respectively. The rest of our benchmark set consists of scenarios that never took place, but were crafted using real liner shipping data to examine how our algorithms scale. Since all of our data in the benchmark is confidential information from our industrial collaborator, we have duplicated the confidential instances and perturbed the costs, revenues, amounts of cargo in demands, and randomly deleted/added demands to create a publicly available benchmark. We combine publicly available liner shipping data, such as ship information and port fees, from the ENERPLAN benchmark [17] with randomly perturbed data from our collaborator. We perturb all values not already contained in the ENERPLAN benchmark by $\pm 20\%$, as in [17], including non-cost/revenue related values such as port productivities, ensuring that no private data is contained in the dataset. We have kept the schedules of the ships in the repositioning scenarios the same, as this is public information. Our public dataset is available at `http://www.decisionoptimizationlab.dk/index.php/datasets/17-research/datasets/59-lsfrpcf`.

This dataset differs from the one in Chapter 5 in that it contains detailed data on container demands, as well as has a stricter time window for when the repositioning is allowed to happen. Thus, the instances in this dataset are not directly applicable to the NCLSFRP.

Tables 6.3a and 6.3b give information about the instances in the confidential and public datasets, respectively. The table gives the instance ID, the number of ships, $|S|$, the number of nodes in the graph, $|V|$, the number of inflexible arcs, $|A^i|$, the number of flexible arcs, $|A^f|$, the number of demands, $|M|$, the number of ports with equipment surpluses or demands, $|E| = |\cup_{q \in Q} V^{q^*}|$, and finally the number of SOS opportunities, $|SOS|$. The main difference between the confidential and public instances is their cost structure, as well as the number of demands and amount of cargo in each demand. The instances range in size from 3 to 11 ships with various SOS and equipment opportunities made available in each instance. Our instances have between 30 and 379 nodes, and up to 11979 arcs. The number of demands can be as high as 1748, although most instances have less than 400 demands.

ID	$\|S\|$	$\|V\|$	$\|A^i\|$	$\|A^f\|$	$\|M\|$	$\|E\|$	$\|SOS\|$
repo1c	3	30	94	0	27	0	1
repo2c	3	30	94	0	27	0	2
repo3c	3	37	113	0	25	0	2
repo4c	3	40	143	0	21	0	3
repo5c	3	47	208	0	24	0	3
repo6c	3	47	208	0	24	0	3
repo7c	3	53	146	0	51	0	4
repo8c	3	104	1015	121	67	6	3
repo9c	3	104	1015	121	67	9	3
repo10c	4	58	389	0	150	0	0
repo11c	4	62	389	40	150	6	0
repo12c	4	74	469	0	174	0	2
repo13c	4	80	492	0	186	0	4
repo14c	4	80	492	0	186	24	4
repo15c	5	68	237	0	214	0	0
repo16c	5	103	296	0	396	0	5
repo17c	6	100	955	0	85	0	0
repo18c	6	133	1138	0	101	0	9
repo19c	6	133	1138	0	101	33	9
repo20c	6	140	1558	0	97	0	4
repo21c	6	140	1558	0	97	13	4
repo22c	6	140	1558	0	97	37	4
repo23c	6	152	1597	162	103	71	9
repo24c	7	75	395	0	196	0	3
repo25c	7	77	406	0	195	0	0
repo26c	7	77	406	0	195	16	0
repo27c	7	78	502	0	237	0	0
repo28c	7	89	537	0	241	0	4
repo29c	7	89	537	0	241	19	4
repo30c	8	126	1154	0	165	0	0
repo31c	8	126	1300	0	312	0	0
repo32c	8	140	1262	0	188	0	3
repo33c	8	209	2211	453	213	50	3
repo34c	9	304	9863	0	435	0	0
repo35c	9	357	11289	38	1075	118	4
repo36c	9	364	11078	0	1280	0	4
repo37c	9	371	10416	0	1270	114	7
repo38c	9	373	11979	38	1280	126	4
repo39c	9	379	10660	0	1371	0	7
repo40c	9	379	10660	0	1371	118	7
repo41c	10	249	7654	0	473	0	0
repo42c	11	275	5562	0	1748	0	5
repo43c	11	320	11391	0	1285	0	0
repo44c	11	328	11853	0	1403	0	4

(a) Confidential dataset.

ID	$\|S\|$	$\|V\|$	$\|A^i\|$	$\|A^f\|$	$\|M\|$	$\|E\|$	$\|SOS\|$
repo1p	3	36	150	0	28	0	1
repo2p	3	36	150	0	28	0	2
repo3p	3	38	151	0	24	0	2
repo4p	3	42	185	0	20	0	3
repo5p	3	51	270	0	22	0	3
repo6p	3	51	270	0	22	0	3
repo7p	3	54	196	0	46	0	4
repo8p	3	108	1185	126	50	6	3
repo9p	3	108	1185	126	50	10	3
repo10p	4	58	499	0	125	0	0
repo11p	4	62	499	38	125	6	0
repo12p	4	74	603	0	145	0	2
repo13p	4	80	632	0	155	0	4
repo14p	4	80	632	0	155	24	4
repo15p	5	71	355	0	173	0	0
repo16p	5	106	420	0	320	0	5
repo17p	6	102	1198	0	75	0	0
repo18p	6	135	1439	0	87	0	9
repo19p	6	135	1439	0	87	33	9
repo20p	6	142	1865	0	80	0	4
repo21p	6	142	1865	0	80	13	4
repo22p	6	142	1865	0	80	37	4
repo23p	6	153	1598	159	89	71	9
repo24p	7	75	482	0	154	0	3
repo25p	7	77	496	0	156	0	0
repo26p	7	77	496	0	156	16	0
repo27p	7	79	571	0	188	0	0
repo28p	7	90	618	0	189	0	4
repo29p	7	90	618	0	189	19	4
repo30p	8	126	1450	0	265	0	0
repo31p	8	130	1362	0	152	0	0
repo32p	8	144	1501	0	170	0	3
repo33p	8	212	2227	433	179	50	3
repo34p	9	304	10577	0	344	0	0
repo35p	9	357	11284	35	874	118	4
repo36p	9	364	11972	0	1048	0	4
repo37p	9	371	11371	0	1023	114	7
repo38p	9	373	11972	35	1048	126	4
repo39p	9	379	11666	0	1109	0	7
repo40p	9	379	11666	0	1109	118	7
repo41p	10	249	8051	0	375	0	0
repo42p	11	279	6596	0	1423	0	5
repo43p	11	320	13058	0	1013	0	0
repo44p	11	328	13705	0	1108	0	4

(b) Public dataset.

Table 6.3: Instance information for the confidential and public datasets.

6.7.2 Arc, Node and Path Flow Approach Evaluations

We evaluate the arc flow model on the entire dataset, as well as the node flow approaches on those instances that have no flexible components. Tables 6.4 and 6.5 show the CPU times in seconds of the arc flow model (AF) from Section 6.2, node flow model with equipment modeled as flows (EAF) from Section 6.4.2, and the node flow model with equipment modeled as demands (EAD) from Section 6.4.3 on the confidential and public LSFRP datasets, respectively. Additionally, we report the time required for the path-based model to converge using the arc flow model as its subproblem (PM-AF), the EAF model as its subproblem (PM-EAF) and the EAD model as its subproblem (PM-EAD). Note that the EAD and EAF models can only solve inflexible instances, which are marked with an "I" in the Type column. This restriction also holds for the path-based model where only the AF subproblem is able to handle instances with flexible visitations and arcs.

On the confidential dataset, shown in Table 6.4, the AF model is unable to solve any problem after repo34c, which corresponds to instances with 9 vessels or more, and a graph size of nearly 10,000 arcs. On 8 of the 11 confidential instances that the AF model cannot solve, CPLEX runs out of memory. This happens due to the large number of variables needed to model the flow of demands through the graph. On instances the AF model can solve, the performance varies greatly, with small instances solvable in as little as 0.03 seconds (repo4c), to requiring as much as 2075.82 seconds on repo30c.

Both the EAF and EAD models are able to solve more instances than the AF model within the 5 hour timeout period, with EAF solving 5 more instances and EAD solving 6 more instances. Additionally, for instances with between one and eight ships (repo1c – repo33c), the CPU time is significantly improved, with an average decrease in CPU time of 138 seconds for both EAF and EAD over AF. The largest speed improvement is on instance repo30c, where EAF and EAD are roughly 500 times faster than AF.

Even on the instances where EAD or EAF timeout, CPLEX is able to provide feasible solutions, unlike in the case of the AF model, where no feasible solution is found for any instance that exceeds the memory allotment or CPU timeout. The EAD model is able to find a solution with a 10.16% gap (to the LP relaxation) for repo42c, a 96.41% gap on repo43c, and an 88.96% gap on repo44c. Although EAD requires over an hour to solve several instances, it finds the optimal solution within an hour on both repo37c and repo40c, but requires extra time to prove optimality. On repo39c and repo40c, EAD is able to find an optimality gap of 8.31% and 10.07% within an hour, respectively.

The path-based model offers significant speed improvements over the AF, EAF and EAD models, and using the EAF or EAD models in the subproblem, instances repo42c – repo44c can be solved. This leaves just two instances out of the entire dataset for which we do not yet know the optimal solution. The path based model is only guaranteed to provide a lower bound on the optimal value of the LSFRP. This is a product of the column generation model we solve, which must use an LP to solve the restricted master problem. This means that one or more variables could be provided non-integer

ID	Type	AF	EAF	EAD	PM-AF	PM-EAF	PM-EAD
repo1c	I	0.04	0.48	0.04	0.22	0.06	0.10
repo2c	I	0.04	0.04	0.04	0.12	0.05	0.06
repo3c	I	0.03	0.03	0.03	0.12	0.06	0.07
repo4c	I	0.03	0.37	0.03	0.15	0.08	0.09
repo5c	I	0.05	0.02	0.03	0.15	0.08	0.07
repo6c	I	0.06	0.04	0.03	0.14	0.07	0.07
repo7c	I	0.06	0.04	0.04	0.17	0.08	0.08
repo8c	F	1.92	-	-	3.04	-	-
repo9c	F	1.91	-	-	3.14	-	-
repo10c	I	16.08	0.34	0.33	16.21	0.76	0.72
repo11c	F	14.50	-	-	21.94	-	-
repo12c	I	72.91	0.88	0.86	43.15	1.50	1.27
repo13c	I	231.47	0.87	0.82	42.90	1.36	1.33
repo14c	I	182.06	0.93	1.05	46.94	1.31	1.37
repo15c	I	0.39	0.23	0.23	1.32	0.43	0.42
repo16c	I	0.95	0.36	0.35	2.69	0.56	0.55
repo17c	I	5.41	0.73	0.71	60.36	1.35	1.27
repo18c	I	6.72	0.79	0.75	40.77	1.87	1.69
repo19c	I	5.68	0.87	0.98	49.13	2.97	1.69
repo20c	I	313.55	1.45	1.41	472.24	5.46	5.05
repo21c	I	47.78	1.62	1.55	216.42	5.54	5.04
repo22c	I	39.51	1.19	1.65	339.88	5.39	4.96
repo23c	F	19.79	-	-	71.30	-	-
repo24c	I	2.30	0.35	0.32	2.78	0.30	0.28
repo25c	I	2.69	0.32	0.30	3.26	0.26	0.24
repo26c	I	1.96	0.41	0.34	2.69	0.29	0.25
repo27c	I	94.48	0.53	0.51	313.98	1.48	1.42
repo28c	I	174.55	0.66	0.55	367.98	1.59	1.53
repo29c	I	186.38	0.62	0.59	335.09	1.63	1.55
repo30c	I	2075.82	3.76	4.15	302.02	7.64	7.33
repo31c	I	99.02	4.93	4.76	13223.75	20.73	20.45
repo32c	I	487.93	6.97	7.12	525.26	7.73	7.32
repo33c	F	548.11	-	-	1302.04	-	-
repo34c	I	CPU	2256.99	2532.11	CPU	63.88	57.69
repo35c	F	Mem	-	-	Mem	-	-
repo36c	I	Mem	CPU	16203.26	Mem	127.84	99.08
repo37c	I	Mem	7033.32	6330.48	CPU	152.93	69.58
repo38c	F	Mem	-	-	Mem	-	-
repo39c	I	Mem	7125.89	7142.55	Mem	119.04	105.07
repo40c	I	Mem	15857.61	10049.79	CPU	175.52	103.43
repo41c	I	CPU	2543.09	3954.17	CPU	77.23	70.94
repo42c	I	CPU	CPU	CPU	CPU	4938.54	5886.86
repo43c	I	Mem	CPU	CPU	Mem	5354.41	4343.04
repo44c	I	Mem	CPU	CPU	Mem	5285.74	4859.72

Table 6.4: Time in seconds required to solve the arc flow model (AF), node flow with equipment modeled as flows model (EAF), and the node flow with equipment modeled as demands model (EAD), and the path based model the arc flow model as a sub problem (PM-AF), using equipment as flows for the sub problem (PM-EAF), and equipment as demands for the sub problem (PM-EAD) to optimality in CPLEX 12.4 on the confidential dataset with a 5 hour timeout. Instances are marked as either inflexible (I) or flexible (F).

values. However, the path-based approach finds integer solutions on every instance in our dataset, meaning it finds the optimal solution. We note, however, that there is no theoretical guarantee that this will occur across any arbitrary LSFRP instance.

The two instances the path-based model cannot solve, repo35c and repo38c, both have flexible components. On these problems, the subproblem using the AF model runs out of memory, and the EAD and EAF models cannot be used due to their lack of support for flexible visitations. Both PM-EAD and PM-EAF significantly outperform PM-AF, with the largest speedup on instance repo31c, where the path based model requires over 600 times the CPU time to solve the problem. The path based approach gets stuck in difficult to solve subproblems that provide columns to the master problem with only incrementally better objectives. Interestingly, the AF model only requires 99 seconds to solve repo31c compared to PM-AF's over 13,000 seconds. In fact, PM-AF requires on average 2.14 times the amount of CPU time on flexible instances as AF. AF has an average CPU usage of 144.7 seconds on inflexible instances versus 539.7 seconds for PM-AF. The average performance is heavily weighted by several instances which require hundreds of seconds. However even looking at the geometric mean, PM-AF requires 2.66 times the CPU time of AF, with a geometric mean of 15.35 for PM-AF versus 5.78 for AF. This is mainly due to the slow solution of the subproblem in PM-AF, which happens a number of times over the course of optimization. For future work, a dynamic programming or improved solution approach could be employed to solve the subproblem faster.

On instances repo1c – repo32c, PM-EAF and PM-EAD generally require slightly more CPU time than the EAF or EAD models, with EAD taking on average 1.1 seconds and PM-EAD taking 2.3 seconds, respectively. Additionally, EAF requires 1.1 seconds while PM-EAF takes 2.5 seconds. However, on repo34c – repo41c, PM-EAF and PM-EAD provide significant speedups on the inflexible instances over the EAF and EAD models; with PM-EAD solving these instances in only 84.3 seconds on average versus 7702.1 seconds for EAD. Furthermore, PM-EAD and PM-EAF solve instances repo42c – repo44c in under two hours of CPU time. We can conclude that there the path-based approach is most effective on large instances (at least 9 ships), and should be used in conjunction with the EAD or EAF models for the subproblem, when possible.

The CPU times of the arc flow, node flow and path-based models on the public instances are given in Table 6.5. We see similar performance of the various approaches as on the confidential dataset, as the public instances are, for the most part, very similar to their confidential counterparts. As in the case of the confidential instances that the EAD and EAF approaches greatly outperform the AF model. The performance of PM-AF is also often worse than AF, with repo30p requiring over 20 times the CPU time with PM-AF than with AF. PM-EAD and PM-EAF solve every inflexible instance. For instances above repo30p, PM-EAD and PM-EAF provide order of magnitude speedups on six instances (repo34p – repo42p) over EAD and EAF.

ID	Type	AF	EAF	EAD	PM-AF	PM-EAF	PM-EAD
repo1p	I	0.06	0.08	0.06	0.18	0.10	0.10
repo2p	I	0.06	0.05	0.05	0.16	0.10	0.10
repo3p	I	0.04	0.04	0.04	0.12	0.08	0.07
repo4p	I	0.04	0.23	0.03	0.17	0.08	0.08
repo5p	I	0.07	0.05	0.05	0.16	0.09	0.07
repo6p	I	0.08	0.06	0.05	0.16	0.08	0.07
repo7p	I	0.08	0.05	0.05	0.27	0.13	0.12
repo8p	F	1.89	-	-	2.64	-	-
repo9p	F	1.82	-	-	2.61	-	-
repo10p	I	74.85	0.27	0.26	25.98	0.85	0.81
repo11p	F	38.17	-	-	32.06	-	-
repo12p	I	106.63	0.31	0.28	79.63	1.65	1.59
repo13p	I	99.81	0.33	0.31	99.27	1.78	1.71
repo14p	I	97.15	0.35	0.33	89.34	1.92	1.69
repo15p	I	0.47	0.31	0.29	1.55	0.55	0.53
repo16p	I	1.08	0.39	0.38	2.89	0.68	0.65
repo17p	I	4.64	1.01	0.94	39.42	2.60	2.37
repo18p	I	6.79	0.95	0.82	33.57	1.78	1.59
repo19p	I	8.18	1.03	1.08	31.02	2.82	1.59
repo20p	I	13.84	1.44	1.15	183.35	3.06	2.80
repo21p	I	23.04	2.06	1.36	106.69	3.24	2.84
repo22p	I	17.67	1.69	1.44	117.20	4.77	2.85
repo23p	F	19.58	-	-	47.84	-	-
repo24p	I	2.23	0.37	0.34	3.71	0.45	0.43
repo25p	I	3.19	0.41	0.38	4.88	0.47	0.43
repo26p	I	2.05	0.47	0.40	4.15	0.51	0.44
repo27p	I	1394.44	0.57	0.50	199.79	0.68	0.65
repo28p	I	1099.87	0.55	0.46	97.10	0.61	0.56
repo29p	I	1183.01	0.57	0.54	96.43	0.72	0.57
repo30p	I	307.12	13.10	12.60	7362.09	7.57	7.21
repo31p	I	57.40	30.88	28.89	104.88	3.92	3.69
repo32p	I	65.51	45.99	41.46	210.93	5.37	5.03
repo33p	F	139.99	-	-	384.63	-	-
repo34p	I	CPU	7652.67	7388.75	CPU	48.34	31.16
repo35p	F	CPU	-	-	CPU	-	-
repo36p	I	Mem	CPU	CPU	Mem	92.42	66.10
repo37p	I	Mem	1408.75	790.74	CPU	112.02	53.96
repo38p	F	Mem	-	-	Mem	-	-
repo39p	I	Mem	1701.53	1911.40	Mem	105.02	62.66
repo40p	I	Mem	3178.03	1859.09	Mem	121.14	62.10
repo41p	I	CPU	659.75	727.78	CPU	33.20	31.87
repo42p	I	CPU	4930.85	4006.27	CPU	143.85	133.84
repo43p	I	Mem	CPU	CPU	Mem	2709.54	1870.06
repo44p	I	Mem	CPU	CPU	Mem	2473.23	1646.40

Table 6.5: Time in seconds required to solve the arc flow model (AF), node flow with equipment modeled as flows model (EAF), and the node flow with equipment modeled as demands model (EAD), and the path based model the arc flow model as a sub problem (PM-AF), using equipment as flows for the sub problem (PM-EAF), and equipment as demands for the sub problem (PM-EAD) to optimality in CPLEX 12.4 on the public dataset with a 5 hour timeout. Instances are marked as either inflexible (I) or flexible (F).

6.7.3 SA and LAHC Implementations

We implemented the SA algorithm described in Section 6.5.1 and the LAHC algorithm from Section 6.5.2 in C++11 [71]. The implementations rely on CPLEX 12.4 [69] for computing components of the objective function, as well as Google OR Tools [56] for computing the assignment problem in the DRH heuristic. Our implementation is able to process over 700,000 SA iterations per second on smaller instances, where an iteration is an application of a neighborhood operator to the current solution and an update of the objective function, and 7,100 iterations per second on our largest instance, repo44c.

Parameter Tuning

Both the SA and LAHC algorithms have several parameters which can affect their performance, and in order to ensure a fair comparison of SA and LAHC against the MIP model, we perform parameter tuning. There are many suggestions in the literature for parameter settings of the components of SA algorithms [68, 72]. The LAHC is less well studied, but its primary parameter, the solution list length, also has suggested values in the literature [19]. We have used these guidelines in our parameter tuning procedure, but ultimately rely on the performance of our SA and LAHC approaches on a training set of instances to determine which parameters are the best for the LSFRP. In order to avoid overtuning our algorithm to the instances of the LSFRP that we present, we tune our SA algorithm on a training set of 15 instances from the confidential dataset and validate the performance of the parameters on the entire set of instances. The instances were chosen at random from the confidential instances. The training set consists of 15 instances, which is a little over one third of our dataset. This is a standard amount in the machine learning and parameter tuning literature [7, 74]. The instances comprising our training set are: repo15c, repo17c, repo18c, repo19c, repo21c, repo23c, repo32c, repo35c, repo36c, repo37c, repo38c, repo39c, repo41c, repo42c, and repo43c.

We first tune the SA algorithm, for which there are 13 parameters to tune in total. The feasible shortest paths heuristic has three parameters, γ, ℓ^{Cargo} and ℓ^{Eqp}, which describe the cost factor to use when estimating flexible arc costs, the amount of cargo profit to earn at the origin and destination visitations of a demand, and the amount of equipment to "load" in the heuristic, respectively. The SA algorithm has seven parameters, α, t^{Init}, t^{Min}, r^{Itrs}, r^{Reheat}, β, $r^{Restart}$, which are the geometric temperature decrease factor, the starting SA temperature, the SA convergence temperature, the maximum number of non-improving iterations before convergence, the number of non-improving reheats before convergence, the reheating factor, and the number of reheats before resetting the current solution to the starting solution, respectively. Finally, there are three parameters that control the penalization of infeasible solutions in the objective. The parameters p, p^{PI}, and p^{T} are the penalization of multiple vessels utilizing the same visitation in their paths, the penalty for two vessels using the same phase-in visitation, and the penalty for a vessel's path being temporally infeasible. Note that we penalize phase-in disjointness violations separately from normal violations because they tend to be a more difficult infeasibility for the SA to fix.

Cat.	Param.	Discretized domain values.	DRH	SPH	GH
GH/ SPH	γ	0.0, 0.1, 0.25, 0.5, 0.75	0.25	0.25	0.25
	ℓ^{Cargo}	0.0, 0.1, 0.25, 0.5, 0.75, 0.95	0.95	0.95	0.95
	ℓ^{Eqp}	0, 10, 50, 100, 250, 500, 1000	100	100	100
SA	α	0.997, 0.998, 0.999, 0.9999, 0.99999, 0.999999	0.999999	0.999999	0.999999
	t^{Init}	1×10^4, 5×10^4, 1×10^5, 5×10^5, 1×10^6	1×10^6	1×10^6	5×10^5
	t^{Min}	1×10^{-8}, 1×10^{-10}, 1×10^{-12}, 1×10^{-15}	1×10^{-8}	1×10^{-8}	1×10^{-8}
	r^{Itrs}	500, 1000, 2500, 5000, 7500, 1×10^4	7500	10000	7500
	r^{Reheat}	1, 5, 10, 20	20	20	20
	β	0.1, 0.25, 0.5, 0.75	0.75	0.75	0.75
	$r^{Restart}$	1, 5, 10, 20	10	10	20
Penalty	p	1×10^5, 2×10^5, 5×10^5, 7×10^5, 1×10^6, 2×10^6, 5×10^6, 7×10^6, 1×10^7	7×10^6	7×10^6	1×10^6
	p^{PI}	1×10^5, 2×10^5, 5×10^5, 7×10^5, 1×10^6, 2×10^6, 5×10^6, 7×10^6, 1×10^7	7×10^6	5×10^6	5×10^6
	p^T	1×10^5, 2×10^5, 5×10^5, 7×10^5, 1×10^6, 2×10^6, 5×10^6, 7×10^6, 1×10^7	1×10^5	1×10^5	7×10^5

Table 6.6: The discretized parameter domains used in hand tuning are given with parameters classified into several categories. The best parameter for each initial solution heuristic as determined through parameter tuning are given on the right side of the table.

We tune the SA algorithm by hand, running each parameter configuration ten times on each training instance, each time with a different random seed. Running each parameter configuration multiple times is necessary due to the stochastic nature of the SA algorithm. In order to avoid a combinatorial explosion of parameter settings, we assume that parameters are independent of each other. While this is clearly not a true assumption, it is the only way to perform parameter tuning across so many parameters without spending extraordinary amounts of CPU time. We then choose the best parameter value for each parameter based on the total profit earned by using each parameter.

Table 6.6 gives the discretized parameter domains we used during hand tuning, as well as the best value for each parameter for each initial heuristic with all SA neighborhoods enabled. We determined which parameter was the best by computing the total profit earned by each parameter across the training set.

We perform the same parameter tuning on the LAHC approach as on SA. Table 6.7 gives an overview of the parameters considered for LAHC, along with the best parameters found during hand tuning. We use the greedy initial starting solution proposed like we do for the SA. The parameters γ, ℓ^{Cargo}, and ℓ^{Eqp} control the cost of flexible arcs, the factor of profit earned from cargo, and the factor of profit earned from equipment, respectively, in the greedy initial heuristic. The LAHC parameters ℓ, r^{Itrs}, and $r^{Restart}$ determine the list length, the number of failed iterations before restarting, and the maximum number of restarts, respectively. Finally, the parameters p, p^{PI}, and p^T describe the amount of penalization in the objective for non-node disjoint paths, multiple vessels utilizing the same phase-in slot, and temporal infeasibility within a vessel path, respectively.

Category	Parameter	Discretized domain values.	LAHC
GH	γ	0.0, 0.1, 0.25, 0.5, 0.75	0.25
	ℓ^{Cargo}	0.0, 0.1, 0.25, 0.5, 0.75, 0.95	0.1
	ℓ^{Eqp}	0, 10, 50, 100, 250, 500, 1000	100
LAHC	ℓ	10, 100, 1000, 10000, 100000	10000
	r^{Itrs}	1000, 10000, 100000	10000
	$r^{Restart}$	10, 100, 1000	10
Penalty	p	1×10^5, 2×10^5, 5×10^5, 7×10^5, 1×10^6, 2×10^6, 5×10^6, 7×10^6, 1×10^7	2×10^6
	p^{PI}	1×10^5, 2×10^5, 5×10^5, 7×10^5, 1×10^6, 2×10^6, 5×10^6, 7×10^6, 1×10^7	2×10^6
	p^{T}	1×10^5, 2×10^5, 5×10^5, 7×10^5, 1×10^6, 2×10^6, 5×10^6, 7×10^6, 1×10^7	5×10^6

Table 6.7: The discretized parameter domains used in hand tuning are given with parameters classified into several categories. The best parameter for LAHC is shown on the right.

	DRH	SPH	GH
# Best Obj.	29	30	13
# Worst Obj.	9	13	31
Obj. Average	-4.14×10^5	-1.34×10^6	-8.13×10^6
Obj. Median	-1.93×10^6	-1.93×10^6	-3.76×10^6

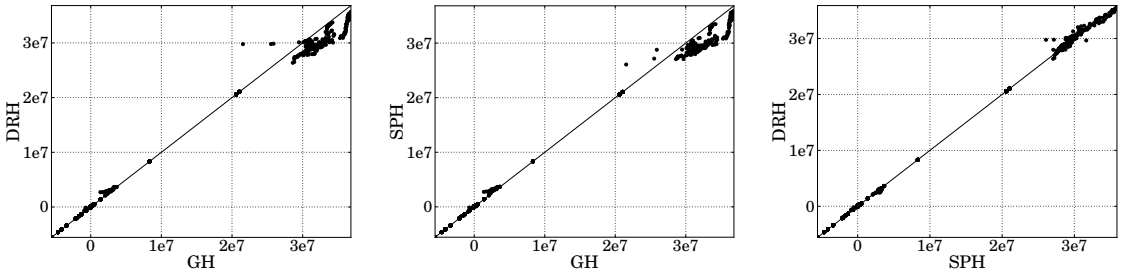
Table 6.8: Starting solution statistics for all three heuristics on the confidential dataset.

6.7.4 Initial Solution Heuristics Comparison

We compare the performance of the initial solution heuristics introduced in Section 6.5.4 across our dataset using the tuned parameters from Section 6.7.3. Table 6.8 describes the performance of the starting heuristics across the dataset.

The DRH and SPH heuristics achieve the best initial objectives out of the three heuristics on 29 and 30 of the instances, respectively (on many instances both heuristics return the same solution), while the GH heuristic only provides the best value on 13 instances. In order to determine the significance of the difference in the means of the solutions, we use a one-way ANOVA test (see, e.g., [141]) with the null hypothesis that the means of SPH, DRH and GH on the private dataset are not different from each other. We are unable to reject the hypothesis given the value $p = 0.581$, which is right on the border of significance. Although we can see that the solutions provided by each heuristic are not exactly the same, we do not have significant enough evidence to say that any one initial solution heuristic is better than another.

Once the SA algorithm finishes, the answers tend to be relatively similar between the various starting heuristics. Figure 6.5 compares the final solutions of the SA algorithm using different initial heuristics. Points below the line $y = x$ mean that the heuristic on the x-axis has better performance, points above the line mean the heuristic on the y-axis has better performance. This speaks to the strength of our SA algorithm, in that it is able to find high quality solutions despite having initial solutions of varying quality. Of particular interest is the fact that SA tends to find better solutions when starting from worse objective values. GH outperforms both SPH and DRH on instances with high objective values.



(a) GH vs. DRH (b) GH vs. SPH (c) SPH vs. DRH

Figure 6.5: Comparisons of the final objective of the SA algorithm with the three initial heuristics on the confidential dataset.

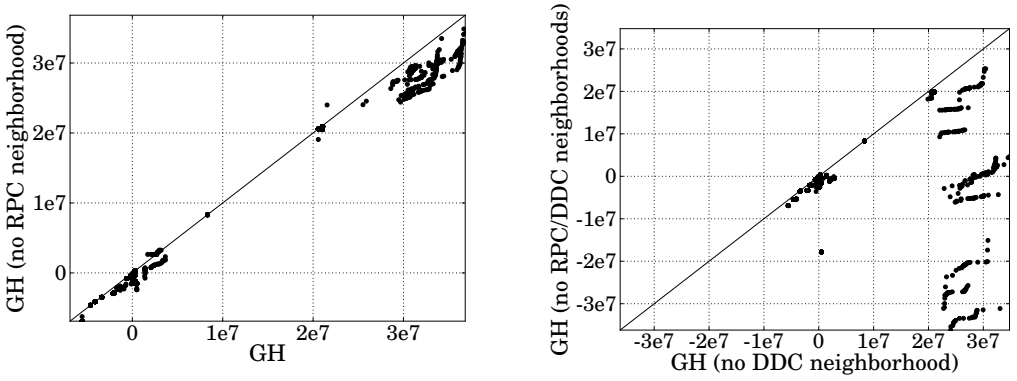
6.7.5 Neighborhood Analysis

We perform an analysis of two of the neighborhoods from Section 6.5.5, the RPC neighborhood and the DDC neighborhood, in order to determine if they are beneficial to the SA algorithm. We do not analyze the visitation addition, removal and swapping neighborhoods because they are basic building blocks that any LS would need to be successful.

Random Path Completion

We tune the parameters of the SA algorithm with and without the RPC neighborhood in order to determine whether the random paths generated are beneficial to the SA. We perform this experiment both with and without the DDC neighborhood, and solve each instance using 25 different seeds. Figures 6.6a and 6.6b show the performance of the SA algorithm using an initial solution generated by GH with the RPC neighborhood vs. not using the RPC neighborhood, both using the DDC neighborhood and without using DDC, respectively. We only show data using the GH initial heuristic since the performance of all three heuristics is similar after optimization. Points below the line $y = x$ in the scatter plots indicate better performance for the RPC neighborhood. The RPC neighborhood's usefulness is clear both with and without the DDC neighborhood. Not using the RPC neighborhood outperforms using the neighborhood only on several instances, and in many cases, the RPC neighborhood is able to find solutions that are orders of magnitude better than without the neighborhood. The line like structures in Figure 6.6b are due to multiple runs of instances that result a number of solutions with similar objectives.

With the DDC neighborhood, the average objective of GH with the RPC neighborhood is 8.6×10^6, versus an average objective of 7.3×10^6 without the RPC neighborhood. A t-test confirms the statistical significance of our findings, with $p < 1 \times 10^{-4}$, allowing us to reject the null hypothesis that the RPC neighborhood does not improve the solution quality. The difference in objective quality becomes even more pronounced when



(a) With DDC neighborhood.

(b) Without DDC neighborhood.

Figure 6.6: Effectiveness of the RPC neighborhood with and without the DDC neighborhood using the GH initial solution heuristic.

the DDC neighborhood is turned off. In this case, using the RPC neighborhood has an average objective of 7.4×10^6, but turning off RPC results in only -1.9×10^6.

We conclude that the RPC neighborhood is an important mechanism for the SA to explore new paths in the graph and avoid getting stuck in a local optimum. In contrast to the visitation addition, removal and swap operators, which can help refine a solution, the RPC neighborhood creates large changes in solutions that are critical to good performance from our SA algorithm.

Demand Destination Completion

We also test the effectiveness of the DDC neighborhood in order to see how much the neighborhood benefits the solution. Figure 6.7 plots the performance of the SA algorithm using initial solutions generated by GH with the DDC neighborhood vs. without the DDC neighborhood on each instance in the confidential dataset with 25 different seeds per instance. We hand tuned parameters for the SA for both with and without the neighborhood for fairness of comparison. Points below the line $y = x$ indicate a higher profit for the DDC neighborhood, whereas points above the line indicate that turning the neighborhood off provides a higher profit. The benefit of the DDC neighborhood is clearly demonstrated by the plot, with the majority of the instances lying below the line. Indeed, a t-test confirms the statistical significance of the result, allowing us to reject the null hypothesis that the mean of the SA performance with the DDC neighborhood is the same as without the neighborhood with a significance level of $p < 1 \times 10^{-4}$. The average objective across all instances and seeds with the neighborhood is 8.6×10^6, and without the neighborhood is 7.4×10^6, an improvement of 14%.

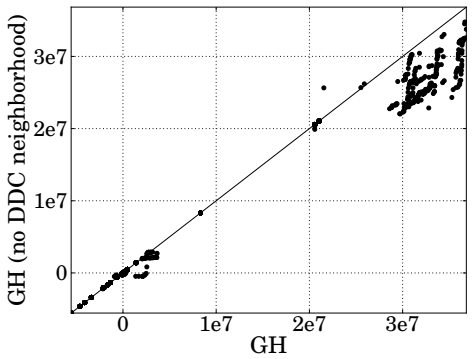


Figure 6.7: Performance of the SA algorithm with and without the DDC neighborhood using the GH initial solution heuristic.

6.7.6 SA and LAHC Results

We ran the SA algorithm with all three starting heuristics, and LAHC with the GH starting heuristic on all instances in our dataset with until the algorithm converges or a 10 minute timeout is reached. Table 6.9 shows the best objective values obtained over 25 runs of each configuration of SA and LAHC on the confidential dataset. For comparison, we provide the optimal objective when it is known and the optimality gap for the heuristic approaches. All objectives are given in tens of thousands. SA is able to find the optimal solution on 29 instances using the SPH and GH initial solution heuristics, and 28 optimal solutions with the DRH approach. LAHC-GH finds 27 optimal solutions. Instance repo9c poses a particular challenge for SA-DRH, although all other methods are able to find optimal solutions on the instance. This is likely due to the simplicity of DRH, which requires local search approaches to add many extra visitations to the solution to find a good objective value. Both SA and LAHC find solutions within 20% of the optimal solution in most cases, with the exception of instance repo34c, where none of the local search approaches are able to find an optimality gap under 52%. The reason for this is not entirely clear, especially since repo34c has no flexible components, equipment or SOS opportunities, meaning several difficult components of the problem are removed. The instance does have relatively high cargo volumes, which could make this instance harder than others, although repo42c, repo43c and repo44c also have many containers that are carried, but both SA and LAHC find solutions of around a 20% gap or less. The best overall performance comes from the SA approach with the GH starting heuristic, which is why we used GH for LAHC as well.

The average optimality gap of LAHC-GH roughly 1.7 times larger than that of SA-GH, however it is still under half the gap of SA-DRH. Although this gap indicates that LAHC-GH is able to perform reasonably well across our dataset, we used a t-test to test the null hypothesis that there is no difference between LAHC-GH and SA-GH. We are able to reject this hypothesis across all instances solved with $p < 1 \times 10^{-4}$. This

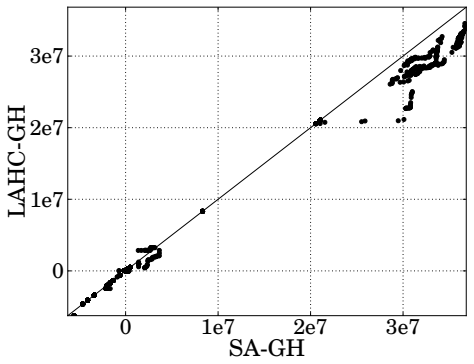


Figure 6.8: Performance of the SA-GH versus LAHC-GH.

provides strong evidence that SA outperforms LAHC on the LSFRP. Indeed, Figure 6.8 shows the performance of SA-GH versus LAHC-GH, where it can be clearly seen that SA-GH outperforms LAHC-GH on nearly every instance in the dataset, across all runs.

Table 6.10 gives the best objective value found for SA and LAHC using varying initial solution heuristics for the public dataset. As in the case of the confidential dataset, LAHC-GH results in an optimality gap twice as high as SA-GH. We are able to reject the null hypothesis that LAHC-GH and SA-GH have the same performance on the public dataset across all instances with $p < 1 \times 10^{-4}$. On the confidential dataset, SA-GH had the lowest average optimality gap, however on the public dataset, SA-SPH's average gap is slightly lower. SA-GH has an average best objective of \$3.471 million versus only \$3.328 million for SA-SPH. Nonetheless, there is no statistically significant difference between using the various initial solution heuristics on the public dataset. Using an ANOVA test across all instances found yields a high p value.

Table 6.11 provides the average objective and optimality gap across all 25 runs of each confidential instance for the SA approach on three initial solution heuristics and on LAHC with the GH heuristic. We also provide the mean of the objectives and optimality gaps across all instances and individual runs. Note that two different means of the optimality gap are provided, one including all gaps ($\oslash$) and one where all gaps over 1.0 are excluded ($\oslash^*$). We provide both averages because several instances with large gaps greatly influence the averages, in such a way as to provide slightly misleading results. For example, the average gap of LAHC-GH is 0.139, which is almost half that of SA-GH, but the performance SA-GH is significantly better than LAHC-GH, as shown by a t-test with $p < 1 \times 10^{-4}$. When we remove these large gaps from both SA-GH and LAHC-GH, we see that the average optimality gap of LAHC-GH is, in fact, almost twice that of SA-GH.

Table 6.12 shows the average objective and optimality gap across all 25 runs of each public instance for the SA approach on three initial solution heuristics and on LAHC with the GH heuristic. The table is structured the same as Table 6.11. Once again, SA is able to solve a number of instances to optimality across all runs that LAHC cannot.

ID	Optimal	SA-DRH		SA-SPH		SA-GH		LAHC-GH	
		Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap
repo1c	-33.91	-33.91	0.000	-33.91	0.000	-33.91	0.000	-33.91	0.000
repo2c	-33.91	-33.91	0.000	-33.91	0.000	-33.91	0.000	-33.91	0.000
repo3c	-55.58	-55.58	0.000	-55.58	0.000	-55.58	0.000	-62.54	0.125
repo4c	-6.30	-6.30	0.000	-6.30	0.000	-6.30	0.000	-6.30	0.000
repo5c	0.44	0.44	0.000	0.44	0.000	0.44	0.000	0.44	0.000
repo6c	0.44	0.44	0.000	0.44	0.000	0.44	0.000	0.44	0.000
repo7c	83.20	83.20	0.000	83.20	0.000	83.20	0.000	83.20	0.000
repo8c	0.44	0.44	0.000	0.44	0.000	0.44	0.000	0.44	0.000
repo9c	0.44	-1.21	1.783	0.44	0.000	0.44	0.000	0.44	0.000
repo10c	205.76	205.76	0.000	205.76	0.000	205.76	0.000	205.76	0.000
repo11c	205.76	205.76	0.000	205.76	0.000	205.76	0.000	205.76	0.000
repo12c	210.34	210.34	0.000	210.34	0.000	210.34	0.000	210.34	0.000
repo13c	210.56	210.56	0.000	210.56	0.000	210.56	0.000	210.56	0.000
repo14c	210.56	210.56	0.000	210.56	0.000	210.56	0.000	210.56	0.000
repo15c	4.91	4.91	0.000	4.91	0.000	4.91	0.000	4.91	0.000
repo16c	4.91	4.91	0.000	4.91	0.000	4.91	0.000	4.91	0.000
repo17c	-16.64	-16.64	0.000	-16.64	0.000	-16.64	0.000	-16.64	0.000
repo18c	-13.38	-13.38	0.000	-13.38	0.000	-13.38	0.000	-13.38	0.000
repo19c	-13.38	-13.38	0.000	-13.38	0.000	-13.38	0.000	-13.38	0.000
repo20c	-20.15	-20.15	0.000	-20.15	0.000	-20.15	0.000	-20.15	0.000
repo21c	-20.15	-20.15	0.000	-20.15	0.000	-20.15	0.000	-20.15	0.000
repo22c	-20.15	-20.15	0.000	-20.15	0.000	-20.15	0.000	-20.15	0.000
repo23c	14.07	14.07	0.000	14.07	0.000	14.07	0.000	12.42	0.117
repo24c	-46.30	-46.30	0.000	-46.30	0.000	-46.30	0.000	-46.30	0.000
repo25c	-41.07	-41.07	0.000	-41.07	0.000	-41.07	0.000	-41.07	0.000
repo26c	-41.07	-41.07	0.000	-41.07	0.000	-41.07	0.000	-41.07	0.000
repo27c	2.89	2.89	0.000	2.89	0.000	2.89	0.000	2.89	0.000
repo28c	2.67	2.67	0.000	2.67	0.000	2.67	0.000	2.67	0.000
repo29c	2.67	2.67	0.000	2.67	0.000	2.67	0.000	2.67	0.000
repo30c	0.62	0.58	0.063	0.58	0.063	0.58	0.063	0.58	0.063
repo31c	3.55	3.45	0.027	3.45	0.027	3.45	0.027	3.45	0.027
repo32c	1.57	1.52	0.031	1.52	0.031	1.52	0.031	1.52	0.031
repo33c	2.38	2.33	0.020	2.33	0.020	2.33	0.020	2.33	0.020
repo34c	77.09	36.48	0.527	37.01	0.520	36.48	0.527	28.61	0.629
repo35c	-	301.33	-	295.71	-	324.01	-	296.93	-
repo36c	354.83	322.79	0.090	313.38	0.117	341.49	0.038	324.49	0.086
repo37c	360.98	337.40	0.065	334.76	0.073	342.23	0.052	327.26	0.093
repo38c	-	316.00	-	310.27	-	344.45	-	291.98	-
repo39c	377.74	351.14	0.070	352.40	0.067	366.81	0.029	345.52	0.085
repo40c	377.74	357.07	0.055	358.92	0.050	368.32	0.025	339.31	0.102
repo41c	34.46	32.68	0.052	32.43	0.059	31.84	0.076	32.40	0.060
repo42c	320.94	307.09	0.043	308.85	0.038	318.07	0.009	278.43	0.132
repo43c	383.02	320.10	0.164	304.87	0.204	332.45	0.132	308.66	0.194
repo44c	383.02	315.14	0.177	306.06	0.201	344.63	0.100	294.93	0.230
$\oslash$	82.76	86.40	0.075	85.47	0.035	89.93	0.027	83.31	0.048
σ	145.98	140.38	0.281	138.79	0.090	146.33	0.083	136.09	0.107

Table 6.9: The best objectives (in tens of thousands) and optimality gaps found with SA versus the optimal objective using all three starting heuristics out of 25 runs on each instance of the confidential dataset.

ID	Optimal	SA-DRH		SA-SPH		SA-GH		LAHC-GH	
		Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap
repo1p	-39.83	-39.83	0.000	-39.83	0.000	-39.83	0.000	-39.83	0.000
repo2p	-39.83	-39.83	0.000	-39.83	0.000	-39.83	0.000	-39.83	0.000
repo3p	-61.77	-61.77	0.000	-61.77	0.000	-61.77	0.000	-61.77	0.000
repo4p	-46.62	-46.62	0.000	-46.62	0.000	-46.62	0.000	-46.62	0.000
repo5p	-8.21	-8.21	0.000	-8.21	0.000	-8.21	0.000	-8.21	0.000
repo6p	-8.21	-8.21	0.000	-8.21	0.000	-8.21	0.000	-8.21	0.000
repo7p	-11.49	-11.49	0.000	-11.49	0.000	-11.49	0.000	-11.49	0.000
repo8p	-8.21	-11.54	0.405	-8.21	0.000	-11.54	0.405	-8.21	0.000
repo9p	-8.21	-11.54	0.405	-8.21	0.000	-12.44	0.514	-8.21	0.000
repo10p	137.61	137.61	0.000	137.61	0.000	137.61	0.000	137.61	0.000
repo11p	137.61	137.61	0.000	137.61	0.000	137.61	0.000	137.61	0.000
repo12p	138.55	138.55	0.000	138.55	0.000	138.55	0.000	138.55	0.000
repo13p	138.86	138.86	0.000	138.86	0.000	138.86	0.000	138.86	0.000
repo14p	138.86	138.86	0.000	138.86	0.000	138.86	0.000	138.86	0.000
repo15p	-36.59	-36.59	0.000	-36.59	0.000	-36.59	0.000	-36.59	0.000
repo16p	-36.59	-36.59	0.000	-36.59	0.000	-36.59	0.000	-36.59	0.000
repo17p	-9.36	-9.36	0.000	-9.36	0.000	-9.36	0.000	-9.97	0.066
repo18p	5.22	5.22	0.000	5.22	0.000	5.22	0.000	5.22	0.000
repo19p	5.22	5.22	0.000	5.22	0.000	5.22	0.000	5.22	0.000
repo20p	-11.85	-11.85	0.000	-11.85	0.000	-11.85	0.000	-13.73	0.159
repo21p	-11.85	-11.85	0.000	-11.85	0.000	-11.85	0.000	-11.85	0.000
repo22p	-11.85	-11.85	0.000	-11.85	0.000	-11.85	0.000	-13.67	0.154
repo23p	5.22	5.22	0.000	5.22	0.000	5.22	0.000	5.22	0.000
repo24p	-53.89	-53.89	0.000	-53.89	0.000	-53.89	0.000	-53.89	0.000
repo25p	-53.13	-53.13	0.000	-53.13	0.000	-53.13	0.000	-53.13	0.000
repo26p	-53.13	-53.13	0.000	-53.13	0.000	-53.13	0.000	-53.13	0.000
repo27p	-28.20	-28.20	0.000	-28.20	0.000	-28.20	0.000	-28.20	0.000
repo28p	-32.13	-32.13	0.000	-32.13	0.000	-32.13	0.000	-32.13	0.000
repo29p	-32.13	-32.13	0.000	-32.13	0.000	-32.13	0.000	-32.13	0.000
repo30p	5.72	4.93	0.138	5.06	0.115	5.35	0.064	2.83	0.505
repo31p	-12.08	-12.08	0.000	-12.08	0.000	-12.08	0.000	-12.08	0.000
repo32p	-10.92	-10.92	0.000	-10.92	0.000	-10.92	0.000	-10.92	0.000
repo33p	-10.92	-10.92	0.000	-10.92	0.000	-10.92	0.000	-11.05	0.011
repo34p	-2.01	-2.01	0.000	-2.01	0.000	-2.01	0.000	-3.86	0.923
repo35p	-	132.01	-	124.98	-	135.82	-	140.08	-
repo36p	160.02	147.67	0.077	148.86	0.070	154.34	0.036	154.92	0.032
repo37p	139.31	129.29	0.072	128.08	0.081	133.84	0.039	134.63	0.034
repo38p	-	142.25	-	143.30	-	158.83	-	160.91	-
repo39p	161.53	146.22	0.095	143.63	0.111	149.44	0.075	151.51	0.062
repo40p	161.53	146.86	0.091	150.18	0.070	153.39	0.050	152.13	0.058
repo41p	-39.60	-46.69	0.179	-51.33	0.296	-43.79	0.106	-58.58	0.479
repo42p	253.60	242.78	0.043	243.28	0.041	244.28	0.037	10.58	0.958
repo43p	223.98	174.68	0.220	183.38	0.181	188.67	0.158	171.24	0.235
repo44p	254.06	175.46	0.309	176.63	0.305	186.70	0.265	172.97	0.319
$\oslash$	33.05	33.11	0.048	33.28	0.030	34.71	0.042	28.52	0.095
σ	91.57	84.44	0.104	84.76	0.072	86.70	0.107	79.93	0.223

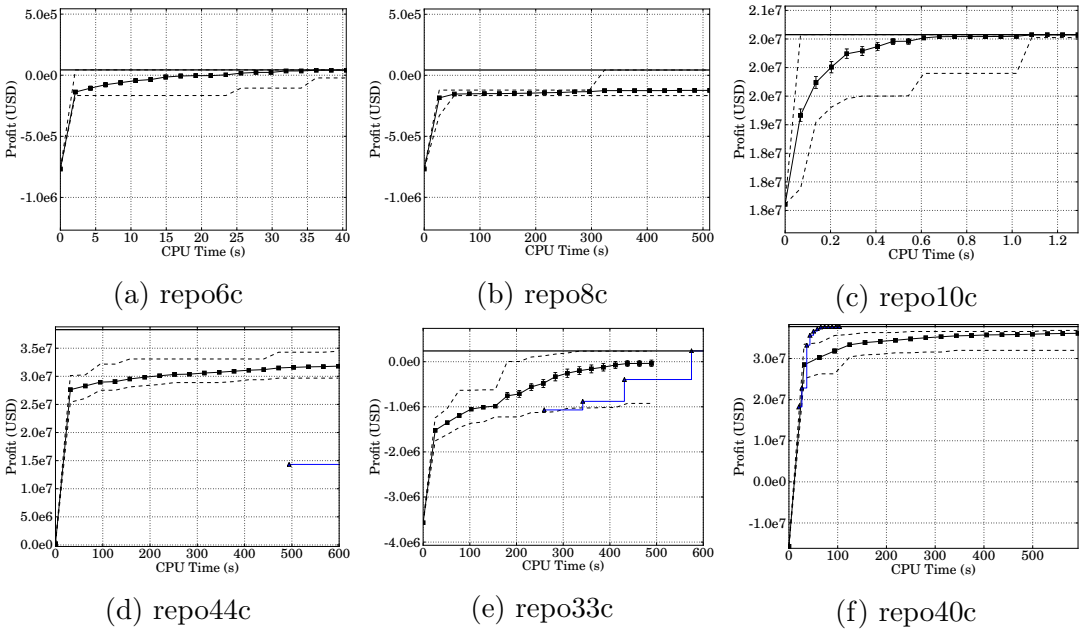
Table 6.10: The best objectives (in tens of thousands) and optimality gaps found with SA versus the optimal objective using all three starting heuristics out of 25 runs on each instance of the public dataset.

		SA-DRH		SA-SPH		SA-GH		LAHC-GH	
ID	Optimal	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap
repo1c	-33.91	-33.91	0.000	-33.91	0.000	-33.91	0.000	-33.91	0.000
repo2c	-33.91	-33.91	0.000	-33.91	0.000	-33.91	0.000	-33.91	0.000
repo3c	-55.58	-55.58	0.000	-55.58	0.000	-55.58	0.000	-62.54	0.125
repo4c	-6.30	-6.30	0.000	-6.30	0.000	-6.30	0.000	-6.30	0.000
repo5c	0.44	0.44	0.000	0.44	0.000	0.44	0.000	0.44	0.000
repo6c	0.44	0.44	0.000	0.44	0.000	0.44	0.000	0.44	0.000
repo7c	83.20	83.20	0.000	83.20	0.000	83.20	0.000	83.20	0.000
repo8c	0.44	-1.12	3.563	-1.20	3.755	-1.22	3.797	0.44	0.000
repo9c	0.44	-1.30	3.989	-1.07	3.453	-1.17	3.687	0.44	0.000
repo10c	205.76	205.76	0.000	205.76	0.000	205.76	0.000	205.74	0.000
repo11c	205.76	205.76	0.000	205.76	0.000	205.76	0.000	205.76	0.000
repo12c	210.34	210.34	0.000	210.34	0.000	210.34	0.000	210.09	0.001
repo13c	210.56	210.56	0.000	210.56	0.000	210.56	0.000	210.56	0.000
repo14c	210.56	210.56	0.000	210.56	0.000	210.56	0.000	210.53	0.000
repo15c	4.91	4.91	0.000	4.91	0.000	4.91	0.000	4.91	0.000
repo16c	4.91	4.91	0.000	4.91	0.000	4.91	0.000	4.91	0.000
repo17c	-16.64	-16.64	0.000	-16.64	0.000	-16.64	0.000	-17.14	0.030
repo18c	-13.38	-13.38	0.000	-13.38	0.000	-13.38	0.000	-13.38	0.000
repo19c	-13.38	-13.38	0.000	-13.38	0.000	-13.38	0.000	-13.38	0.000
repo20c	-20.15	-20.49	0.017	-20.57	0.020	-20.59	0.022	-23.04	0.143
repo21c	-20.15	-20.59	0.022	-20.52	0.018	-20.53	0.019	-22.83	0.133
repo22c	-20.15	-20.49	0.017	-20.46	0.015	-20.58	0.021	-23.45	0.164
repo23c	14.07	14.07	0.000	14.07	0.000	14.07	0.000	9.60	0.318
repo24c	-46.30	-46.30	0.000	-46.30	0.000	-46.30	0.000	-46.30	0.000
repo25c	-41.07	-41.07	0.000	-41.07	0.000	-41.07	0.000	-41.07	0.000
repo26c	-41.07	-41.07	0.000	-41.07	0.000	-41.07	0.000	-41.07	0.000
repo27c	2.89	2.89	0.000	2.89	0.000	2.89	0.000	2.89	0.000
repo28c	2.67	2.67	0.000	2.67	0.000	2.67	0.000	2.67	0.000
repo29c	2.67	2.67	0.000	2.67	0.000	2.67	0.000	2.67	0.000
repo30c	0.62	0.58	0.063	0.58	0.063	0.58	0.063	0.07	0.895
repo31c	3.55	2.87	0.193	2.42	0.318	2.55	0.281	1.13	0.681
repo32c	1.57	1.51	0.037	1.52	0.031	1.51	0.040	1.13	0.281
repo33c	2.38	0.80	0.666	0.56	0.764	-0.37	1.157	0.61	0.746
repo34c	77.09	29.46	0.618	31.32	0.594	28.31	0.633	15.44	0.800
repo35c	-	279.37	-	281.57	-	304.29	-	273.37	-
repo36c	354.83	303.18	0.146	299.63	0.156	318.52	0.102	298.24	0.159
repo37c	360.98	328.85	0.089	328.97	0.089	336.20	0.069	301.44	0.165
repo38c	-	297.22	-	294.62	-	320.20	-	282.65	-
repo39c	377.74	323.20	0.144	324.16	0.142	359.15	0.049	323.23	0.144
repo40c	377.74	338.04	0.105	341.41	0.096	361.31	0.044	324.16	0.142
repo41c	34.46	29.91	0.132	29.98	0.130	25.58	0.258	30.34	0.120
repo42c	320.94	303.45	0.055	295.75	0.078	298.89	0.069	230.38	0.282
repo43c	383.02	291.79	0.238	289.50	0.244	315.95	0.175	296.23	0.227
repo44c	383.02	291.02	0.240	288.56	0.247	318.23	0.169	280.28	0.268
$\oslash$	82.76	291.02	0.246	288.56	0.243	318.23	0.254	280.28	0.139
$\oslash^*$	82.76	291.02	0.059	288.56	0.060	318.23	0.059	280.28	0.111
σ	145.98	7.36	0.841	8.78	0.837	12.45	0.870	7.04	0.337

Table 6.11: The average objectives (in tens of thousands) and average optimality gaps found with SA versus the optimal objective using all three starting heuristics out of 25 runs on each instance of the confidential dataset.

		SA-DRH		SA-SPH		SA-GH		LAHC-GH	
ID	Optimal	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap
repo1p	-39.83	-39.83	0.000	-39.83	0.000	-39.83	0.000	-39.83	0.000
repo2p	-39.83	-39.83	0.000	-39.83	0.000	-39.83	0.000	-39.83	0.000
repo3p	-61.77	-61.77	0.000	-61.77	0.000	-61.77	0.000	-61.77	0.000
repo4p	-46.62	-46.62	0.000	-46.62	0.000	-46.62	0.000	-46.62	0.000
repo5p	-8.21	-9.28	0.130	-9.36	0.140	-8.98	0.094	-8.21	0.000
repo6p	-8.21	-9.39	0.143	-9.15	0.115	-8.65	0.053	-8.21	0.000
repo7p	-11.49	-11.49	0.000	-11.49	0.000	-11.49	0.000	-11.49	0.000
repo8p	-8.21	-12.22	0.488	-12.10	0.473	-12.36	0.506	-8.21	0.000
repo9p	-8.21	-12.36	0.506	-11.98	0.459	-12.44	0.514	-8.21	0.000
repo10p	137.61	137.61	0.000	137.61	0.000	137.61	0.000	137.61	0.000
repo11p	137.61	137.61	0.000	137.61	0.000	137.61	0.000	137.61	0.000
repo12p	138.55	138.55	0.000	138.55	0.000	138.55	0.000	138.55	0.000
repo13p	138.86	138.86	0.000	138.86	0.000	138.86	0.000	138.86	0.000
repo14p	138.86	138.86	0.000	138.86	0.000	138.86	0.000	138.86	0.000
repo15p	-36.59	-36.71	0.003	-36.59	0.000	-37.73	0.031	-37.43	0.023
repo16p	-36.59	-36.64	0.001	-36.59	0.000	-38.11	0.041	-36.59	0.000
repo17p	-9.36	-9.41	0.006	-9.36	0.001	-9.37	0.001	-10.39	0.110
repo18p	5.22	5.22	0.000	5.22	0.000	5.22	0.000	-0.44	0.486
repo19p	5.22	5.22	0.000	5.22	0.000	5.22	0.000	0.46	0.398
repo20p	-11.85	-12.94	0.092	-13.00	0.098	-12.64	0.067	-14.37	0.213
repo21p	-11.85	-12.61	0.065	-12.25	0.034	-13.13	0.108	-14.29	0.207
repo22p	-11.85	-12.97	0.095	-12.82	0.083	-12.50	0.056	-14.34	0.210
repo23p	5.22	5.22	0.000	5.22	0.000	5.22	0.000	3.94	0.245
repo24p	-53.89	-53.89	0.000	-53.89	0.000	-53.89	0.000	-53.89	0.000
repo25p	-53.13	-53.13	0.000	-53.13	0.000	-53.13	0.000	-53.13	0.000
repo26p	-53.13	-53.13	0.000	-53.13	0.000	-53.13	0.000	-53.13	0.000
repo27p	-28.20	-28.20	0.000	-28.20	0.000	-28.20	0.000	-28.20	0.000
repo28p	-32.13	-32.13	0.000	-32.13	0.000	-32.13	0.000	-32.13	0.000
repo29p	-32.13	-32.13	0.000	-32.13	0.000	-32.13	0.000	-32.13	0.000
repo30p	5.72	3.63	0.366	3.90	0.318	3.70	0.353	2.14	0.626
repo31p	-12.08	-12.08	0.000	-12.08	0.000	-12.08	0.000	-12.19	0.009
repo32p	-10.92	-10.92	0.000	-10.92	0.000	-10.92	0.000	-11.09	0.015
repo33p	-10.92	-11.24	0.029	-11.32	0.037	-11.17	0.023	-11.38	0.042
repo34p	-2.01	-2.41	0.203	-2.58	0.288	-2.48	0.237	-6.20	2.091
repo35p	-	119.71	-	117.88	-	131.26	-	137.27	-
repo36p	160.02	140.67	0.121	142.39	0.110	149.68	0.065	150.68	0.058
repo37p	139.31	123.91	0.111	123.37	0.114	129.98	0.067	131.72	0.054
repo38p	-	135.16	-	135.84	-	150.77	-	157.54	-
repo39p	161.53	140.13	0.132	139.29	0.138	146.33	0.094	147.17	0.089
repo40p	161.53	140.28	0.132	140.94	0.127	149.23	0.076	146.28	0.094
repo41p	-39.60	-53.11	0.341	-54.01	0.364	-52.65	0.329	-64.31	0.624
repo42p	253.60	236.07	0.069	230.07	0.093	232.14	0.085	-51.27	0.794
repo43p	223.98	154.12	0.312	150.97	0.326	164.15	0.267	155.80	0.304
repo44p	254.06	160.93	0.367	160.92	0.367	168.39	0.337	154.93	0.390
$\oslash$	33.05	160.93	0.088	160.92	0.088	168.39	0.081	154.93	0.169
$\oslash^*$	33.05	160.93	0.088	160.92	0.088	168.39	0.081	154.93	0.122
σ	91.57	8.34	0.152	9.17	0.157	8.40	0.151	10.28	0.380

Table 6.12: The average objectives (in tens of thousands) and average optimality gaps found with SA versus the optimal objective using all three starting heuristics out of 25 runs on each instance of the public dataset.



(a) repo6c (b) repo8c (c) repo10c

(d) repo44c (e) repo33c (f) repo40c

Figure 6.9: The average performance of SA-GH as it solves several confidential instances.

Figure 6.9 shows the average performance of SA-GH when solving several instances from the confidential dataset. The solid horizontal line shows the optimal objective value for each instance. The error bars provide the standard error across the 25 runs on each instance, and the dashed lines provide the solution value of the best and worst solutions from all runs. On the small instances repo6c, repo8c, and repo10c, the performance of SA-GH is mixed. Although it is able to quickly find the optimal solution for repo10c, SA-GH is not competitive with the EAD or EAF approaches on repo6c or repo8c, as even after several seconds the optimal solution is not found. On instances repo44c, repo33c and repo40c we show the solutions found by PM-EAD as a blue line with triangles. On repo44c and repo33c, SA-GH moves towards the optimal solution more quickly than PM-EAD. However, on repo40c, PM-EAD roughly matches the performance of SA-GH for the first 40 seconds or so, and then exceeds SA-GH, finding the optimal solution in around 100 seconds, whereas SA-GH does not find the optimal solution at all. These results show that there may be benefits to allowing SA-GH to run for several seconds before the path model begins optimizing in some situations. We save this for future work.

6.7.7 Reference Instance Performance

We foresee the approaches presented in this chapter for solving the LSFRP as being used in a decision support system for the LSFRP. In order to test whether or not they are effective at solving real problems, we compare the results of our algorithms with a reference scenario from our industrial collaborator. The scenario, instance repo42c,

encompasses 11 vessels originating from 3 initial services. The vessels seek to create a new service that visits the east coast of South America, Spain and the Middle East. The vessels have a single SOS each week that can be used from Tanjung Pelepas, Malaysia, to Jebel Ali, United Arab Emirates.

Since the reference solution to the scenario faced by our industrial partner was created in advance of the repositioning happening (as one would expect), the people who made it were at a disadvantage compared to our algorithm, which has a more complete view of the opportunities available during the full repositioning period. In order to counter act this unfairness, we calculate the profit of the reference solution under varying relaxations of restrictions present in our model.

Figure 6.10a shows the total profit earned by the reference solution as the size of the *demand delivery window* is increased. This window determines what visitations may be used to deliver cargo. Our real-world data only specifies the date when demands were delivered, not the deadline for delivery. Thus, in our model, we use a value of ± 3 days for the demand delivery window, which means that any visitation within three days of the delivery date is used as a feasible delivery visitation for a demand. By relaxing this demand window to larger values, we allow the reference solution more flexibility as to where cargo gets delivered, which counter-acts the uncertainty planners had when creating the solution. The reference solution profit peaks at \$18,137,488 with a 14 day delivery window.

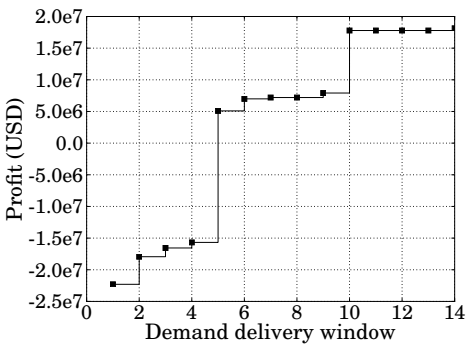
Figures 6.10b and 6.10c show the profit of the incumbent solution of the SA algorithm using the initial solution generated by GH in terms of the number of SA iterations and the CPU time, respectively. Error bars display the standard error across all 25 runs of the instance. The solid blue line shows the profit of the reference solution with a 14 day demand delivery window, and the solid black line the optimal value for the instance. SA-GH is able to find a solution with a better objective than the reference solution, even with a 14 day demand delivery window, in only 20 seconds or so of run time, and only 150 iterations[4]. Figure 6.10d shows the performance of PM-EAD. While it takes longer for PM-EAD to provide a solution that is better than the reference solution than SA-GH, it is able to find an optimal solution in around 1.75 hours of execution time.

6.8 Chapter Summary

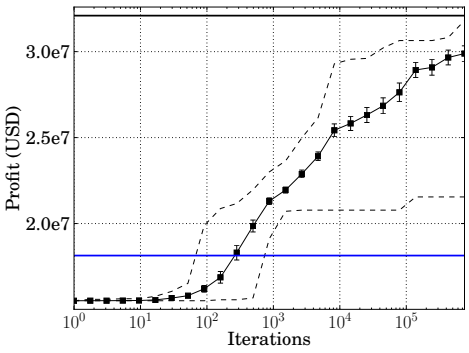
In this chapter, we presented a novel model of an important real-world problem, the LSFRP with cargo flows, and solved it using three different styles of MIP models: an arc flow, a node flow and a path flow model. We also solved the LSFRP with cargo flows using SA and LAHC approaches that harness several initial solution methods and five different neighborhood operators.

Our model takes into account all of the key aspects of the LSFRP, including liner shipping service construction constraints, cargo flows, empty equipment repositioning, cabotage restrictions, sail-on-service opportunities, and maximizes the profit earned

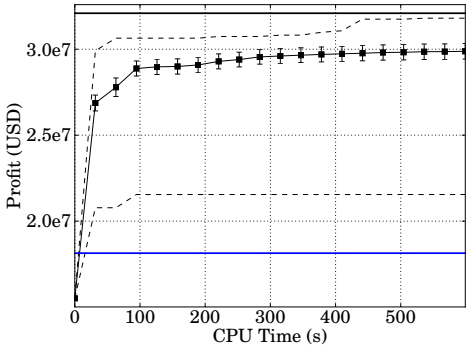
[4]Each iteration represents the evaluation of the objective function.



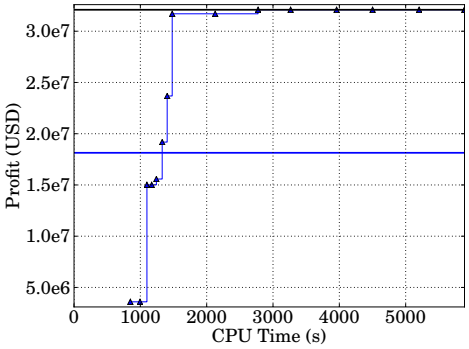
(a) Reference solution objective.



(b) SA-GH profit versus iterations.



(c) SA-GH profit versus time.



(d) PM-EAD profit versus time.

Figure 6.10: The reference solution under varying demand delivery windows, and the performance of SA-GH and PM-EAD on the reference instance, repo42c.

during repositioning. We evaluated our optimal and heuristic approaches, showing that not only does the SA scale to real-world sized problems, but it is also able to find a solution with a significantly higher profit than that of the reference solution from our industrial collaborator.

Our modeling techniques, especially our graph construction, could be applicable to other liner shipping problems, such as an extension to the Vessel Schedule Recovery Problem [18] if SOS opportunities and flexible visitations were included. Additionally, our SA and LAHC approaches, in particular our demand destination completion heuristic and initial solution heuristics, could be useful in other combined multi-commodity flow with vehicle path problems, in which a multi-commodity flow is directed through a graph by the paths of vehicles.

For future work, we intend to integrate the LSFRP into other liner shipping problems, such as fleet deployment and network design, so that these problems can take into account repositioning when making strategic decisions for liner shippers. Additionally, we intend to develop a decision support system that allows user interaction during the repositioning plan creation process.

Chapter 7

Inter-Terminal Transportation

Around the world, ever larger ports are being constructed to keep up with the growth of containerized shipping. Ports routinely contain multiple terminals serving container ships, railways, barges and other forms of hinterland transportation. Containers are often transferred between terminals when they are transshipped between different modes of transportation. The movement of containers between terminals, which is called *inter-terminal transportation* (ITT), represents not only an operational problem for port authorities and terminal operators to deal with, but also a strategic one to be considered during the planning of new terminals and container ports.

The correct choice of the layout of terminals and the transportation connections between them, as well as vehicle type and the number of vehicles, represent expensive and critical decisions that ports must make. The goal of an efficient ITT system is to minimize the delay of containers moving between terminals, so as to reduce and, ideally, eliminate the delayed departure of containers. To this end, we introduce an optimization model based on a time-space graph to determine optimal flows of vehicles and containers in ITT scenarios in order to assist port authorities in their decision making process.

We use an abstract view of ITT operations using a time-space graph with maximum arc capacities and node throughput to model vehicles as flows through the network with transportation demands given as a multi-commodity flow. We focus on minimizing the overall delay experienced by containers, an important consideration for port planners, as the costs of delaying outgoing shipments are usually very high.

Previous work in the area of strategic analysis of ITT primarily deals with simulating inter-terminal operations at the Maasvlakte area of the port of Rotterdam and analyzing the resulting delay of the pickup and delivery of containers ([110, 41, 111]). In contrast to this work, we optimize the flows of containers through the network in order to provide port planners with a better estimation of the cost of using particular vehicles, roadway designs, new infrastructure or traffic planning. This chapter provides the following novel contributions[1], also presented in [152]:

1. The first fully defined mathematical model of ITT.
2. Two exact approaches for minimizing ITT delivery delay.

[1]We thank Pengdi Liu and Nicolas Wind for corrections to the mathematical model.

3. Congestion modeling in the setting of vehicles servicing a multi-commodity flow.

7.1 Problem Description

ITT involves the movement of containers between terminals in a port. There are several types of terminals, including waterside terminals that have a quay where container ships and barges can dock and transfer containers, rail terminals where containers can be loaded onto rail cars, as well as hinterland terminals which can be set far inland and deal with barge, rail or truck transportation. ITT traffic generally consists of either sea-to-sea transportation, i.e., containers being transshipped between vessels, or land-to-sea/sea-to-land transportation, in which containers originating from overseas (the hinterland) are carried to (from) the hinterland by another mode of transportation such as a barge or train.

At first glance, ITT might seem avoidable, either through scheduling container vessels that will transship containers to arrive at the same terminal, or by placing key logistics components of a port all in the same location. However, in nearly every middle to large sized port some amount of ITT is required, due to the fact that avoiding ITT would involve building rail, barge, and container ship connections all in one place, and there simply is not enough space.

There are, therefore, two important problems within the topic of ITT. The first is the purely operational problem of dispatching and routing vehicles to move containers between terminals on a day to day basis in an already constructed port. The second problem is a strategic planning problem for new ports and the expansion of existing ports, which involves several key questions:

1. Is the planned infrastructure sufficient to handle ITT forecasts?
2. What types of vehicles and how many of them are necessary to handle ITT containers?
3. What kind of delays will be experienced, on average, given a particular infrastructure and vehicle configuration?

In this chapter, we provide an optimization model that assists in answering these questions, as well as supports port and terminal authorities in examining the impact of new infrastructure, such as tunnels or bridges, on the overall delay experienced by ITT containers. Thus, while we primarily address the strategic planning issues, our model is also capable of dispatching and routing vehicles in the operational problem at a high level.

7.1.1 Vehicle Types

We consider a range of types of vehicles for ITT that each comes with pros and cons that must be evaluated by decision makers.

Automated Guided Vehicles (AGV)

AGVs are driverless vehicles that can carry up to one forty-foot container or two twenty-foot containers, and have no lifting capabilities of their own. This means that AGVs require cranes for (un)loading operations. Current AGV systems are only allowed in areas where there are no humans in order to prevent accidents. However, this is likely to change as safer AGVs are developed.

Automated Lift Vehicles (ALV)

ALVs, like AGVs, are also driverless vehicles that can carry two twenty-foot containers or one forty-foot container. As their name implies, ALVs have lifting capabilities and do not require external assistance to transport containers. This makes ALVs significantly more versatile than AGVs. However, they generally travel slower.

Multi-Trailer System (MTS)

MTSs consist of several container carrying trailers, that can generally transport up to five 40-foot containers. MTSs require cranes to load them as in the case of AGVs. MTSs are not automated and require a human to drive a tractor unit that pulls the trailer. While this allows more flexibility in the places an MTS can travel, the coupling time of the tractor unit to the trailer can result in a slower turn-around time for the vehicles than AGVs or ALVs. This process is described in detail in [41].

Barges

Barges can be used to transport large quantities of containers between terminals all at once and are driven by humans. Barges are loaded slowly and travel slowly, but have an advantage over road vehicles in that waterways tend to offer shorter connecting distances between terminals than roads, as well as being less congested. Additionally, barges have high capacities in comparison to land based vehicles, and are generally able to carry 40 to 50 containers.

7.1.2 Infrastructure

In order to solve the steep logistical challenges of ITT as container volumes around the world substantially increase, new infrastructure ideas must be considered. The construction of ropeways, monorails, dedicated lanes, tunnels and bridges to connect ports to shunting yards/hinterland logistics centers or to avoid bottlenecks could provide answers for effective ITT. For example, the cost of tunnels and ropeways were considered for connecting the port of Hamburg to hinterland transportation depots in [37]. Although ropeways using current engineering technology were found to be infeasible to carry the weight of fully loaded containers, it shows that with new ideas come new

challenges for evaluating their effectiveness. Our goal is to be able to take any potential infrastructure change into account in a general model.

7.2 Literature Review

Copious studies simulate and optimize container movements within container ports and terminals; see [139, 137]. A particular focus has been placed on intra-terminal simulation and optimization (see [6] for an overview), considering primarily AGV and ALV dispatching and routing (e.g., [16, 57, 106]). Intra-terminal transportation is characterized by its short distances and lack of external traffic interaction. This stands in sharp contrast to ITT, in which vehicles may travel several kilometers to deliver containers over publicly accessible roads. Intra-terminal transportation models and simulations are, therefore, usually not applicable to ITT. Transshipment has been considered in an intra-terminal context in [90]. However, the network based model presented does not take a flow based view of vehicles, meaning modeling congestion is not possible in this framework.

The most relevant works to ITT are [41, 110, 111], which all describe simulation approaches to ITT at the port of Rotterdam. The goal of these studies is to measure the amount of *non-performance*, i.e., the number of containers arriving at their destination terminal after the due time. While a simulation approach is able to model many details of ITT operations, such as loading and unloading procedures and the usage of manned traction units in multi-trailer systems, the approach does not perform optimization other than to tune the parameters by which the optimization is run to try to reduce non-performance. In particular, [111] consider more than just ITT in the simulation approach, including also quay-side movement of containers from ships into stacks. Our model stands in contrast to these approaches in that we optimize ITT using network flows, providing ports with a different view of non-performance.

7.3 Mathematical Model

We model ITT on a time-space graph which has several specially designed structures in order to model traffic congestion as well as the loading and unloading of vehicles. We have created a general model that can incorporate essentially any type of vehicle used for ITT, as well as different types of infrastructure. Our graph uses a carefully designed structure in order to model the handling of slow loading vehicles like barges, and includes components capable of modeling traffic congestion.

We base our model on several key assumptions. The first assumption is that different types of vehicles do not interact with each other except through the loading and unloading of containers at terminals. This means that graph arcs do not have multiple vehicle types traveling on them. This assumption greatly reduces the size of the model, reducing both the number of nodes and arcs considerably. Multiple graph arcs would be otherwise necessary due to the varying speeds, load times, and congestion

properties of vehicles. Second, we assume that ITT containers should be penalized for late arrival, but being early is allowed. This stands in contrast to [41], but we consider containers that are delivered early to be the responsibility of intra-terminal operations. Such containers can be stored either in a stack in the yard or in a rolling buffer. Third, we consider all types of containers as requiring a single unit of capacity on vehicles. In practice this is not true since there are 20 and 40 foot containers, as well as out-of-gauge containers that must be transported between terminals. Our model is capable of handling such containers with minor changes, but we save these for future work. Finally, we abstract away short vehicle activities, such as connecting a tractor to a trailer loaded with containers in an MTS. While many short activities add up over time, modeling them in a network flow requires too fine a discretization. However, due to the short length of such activities, excluding them from the model does not represent a major source of error.

7.3.1 Graph Construction

We first consider a *base graph* which is a non-temporal graph that describes the basic connections between terminals. Let n be the number of terminal nodes and m be the number of intersection nodes. Thus, the base graph $G = (V, A)$ where $V = \{1, \ldots, n + m\}$ is the set of all nodes and A is the set of arcs (i, j) where $i, j \in V$.

Using the base graph, we can now construct a time-space graph containing the special structures that are necessary to model ITT. Let τ be the number of time periods. Let $G^T = (V^T, A^T)$ be the time-space graph where V^T is the set of nodes and A^T the set of arcs. The time-space graph consists of three types of nodes: terminal nodes, intersection nodes, and long-term nodes, or *LT nodes*. LT nodes are copies of the terminal nodes, and model the long term loading/unloading of containers. We require LT nodes because some vehicles, such as trains and barges, take more than a single time discretization to fully (un)load. Without these extra nodes, these vehicles would be able to load and unload at an unrealistic rate unless the model kept track of the remaining capacity of each vehicle at each time step. LT nodes prevent the problem size from being dependent on the number of vehicles in the problem. Let $V^T = \{1, \ldots, \tau(\beta n + m)\}$, where β is a parameter that equals 2 if there are long-term vehicles, such as barges, present in the model, and 1 otherwise. In other words, when LT nodes are needed we create a time-space graph with two nodes in each time period for all terminals with a single node in each period for intersections, and when LT nodes are not needed we create a single node in each time period for both terminals and intersection nodes.

We first show how stationary arcs connect time-space nodes at the same terminal from different periods, then describe LT nodes and their connection to other nodes in the graph, portray how congestion is handled in our graph, discuss the properties of arcs and nodes in the graph, and finally explain the graph's demand structure.

Stationary Arcs

In order to allow vehicles and containers to remain in the same place across time periods, we introduce stationary arcs that connect each terminal, intersection and LT node in subsequent periods. Let A^S be the set of stationary arcs defined as

$$A^S = \bigcup_{0 \le t < (\tau - 1)} \bigcup_{0 \le i < \beta n + m} \{(\tau i + t + 1, \tau i + t + 2)\}.$$

Thus, each node at each time step $(\tau i + t + 1)$ is connected to itself at the next time step $(\tau i + t + 2)$.

Long-Term Loading/Unloading

We introduce an LT node for each time-space terminal node in the graph when there are long-term vehicles, such as barges, present and connect these nodes to each other with an arc in each direction. Let

$$A^{LT} = \bigcup_{0 \le t < \tau} \bigcup_{1 \le i \le n} \{(\tau i + t, \tau n + \tau i + t), (\tau n + \tau i + t, \tau i + t)\},$$

where τ is the number of time periods and n is the number of terminals. Thus, each time-space terminal node has both an incoming and outgoing arc to its associated LT node.

Figure 7.1 shows the relationship between time-space terminal nodes and LT nodes, along with a depiction of the flow of a demand over the LT arcs. The solid lines are stationary arcs (a subset of A^S) for node $i \in V$, the dashed lines represent LT arcs that connect node i to its LT counterpart (a subset of A^{LT}), and the dotted lines are arcs associated with the LT vehicle, either waterways or rail tracks. Solid gray lines show arcs that only road vehicles (such as AGVs, ALVs, and MTSs) may traverse. The figure shows a demand with 12 containers that is being loaded on to an LT vehicle from time period 0 to period 2. Arcs are labeled with the amount of containers being carried on them. The blue arcs show the path of an LT vehicle as it loads the 12 container demand. In each time period, 4 containers are transferred from the terminal node to the LT node until the entire demand is loaded, at which point the LT vehicle can travel to the destination of the demand, or load containers from another demand at some other terminal.

Congestion

Traffic congestion is a key issue facing many ITT systems, as they often utilize roadways open to general traffic. Modeling the basic effects of congestion is an important component of our model. For example, these allow us to assess the effects of rush hour on ITT, the impact of high-occupancy/priority lanes for ITT vehicles in a port area, the effect of tunnels for avoiding traffic, or how an intra-port road network could increase on-time

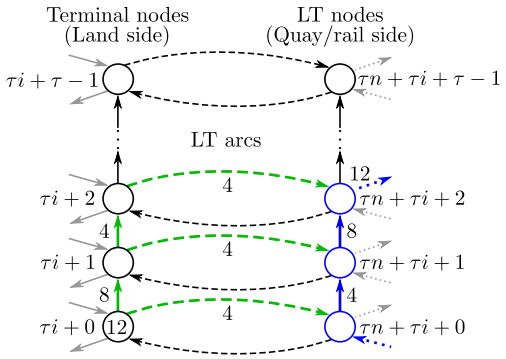


Figure 7.1: Time-space expansion of a terminal node i (left) and its LT node copies (right). A demand originates at node i at time 0 and is slowly loaded over LT arcs (dashed) on to a barge or railcar (blue arcs).

ITT delivery. We therefore impose a capacity on the number of vehicles that can travel on certain arcs, corresponding to the number of vehicles that could be reasonably using the roadway (or waterway) at a particular time, as well as limit the total throughput of intersections in each time period. Congestion is thus modeled as spillover from one time period to another, as stationary arcs have no capacity restrictions. That is, when an arc or intersection is full, vehicles must wait until the next time period to use it. In this way, delay is achieved for overutilized roadways and intersections. We save more detailed congestion models for future work, such as those in [79] or [88], as the details that they model are too fine for the time discretization currently necessary to solve ITT problems. Additionally, neither congestion model is able to take into account our multi-commodity flow or LT nodes, meaning the integration of such techniques into an ITT model is non-trivial.

Arc Properties

Each arc in the time-space graph has a maximum number of vehicles that can travel on it in a single time period, c_{ij}. Let H be the set of all vehicle types, each of which is associated with a maximum container carrying capacity, μ_h. We associate each arc with a specific vehicle type that may travel on the arc, $\eta_{ij} \in H \cup \{\perp\}$, with $(i,j) \in A^T$ and $\perp$ indicates that an arc connects a node to an LT node, as no vehicles utilize such arcs. That is, $\eta_{ij} = \perp$ for all $(i,j) \in A^{LT}$.

We define several functions to assist in accessing the arcs in the model. Let $In(i) = \{j \mid (j,i) \in A^T\}$ ($Out(i) = \{j \mid (i,j) \in A^T\}$) be the set of nodes with arcs to (from) node $i \in V^T$.

Time-space Node Properties

Each time-space node is associated a number of vehicles present at its location at the start of the model, s_i, with $i \in V^T$. In other words, s_i defines the origins of the vehicle

flow. In general, the only nodes with any vehicles before the optimization begins are the terminal nodes at time zero. From a modeling perspective, when the vehicles are made available is irrelevant so we leave this possibility open, even though we do not foresee a scenario in which it would occur.

Each node is capable of performing m_i load or unload moves per time period for road vehicles, and m_i^{LT} moves for barges. Note that although there are multiple types of vehicles in the model, they do not interact at the same nodes, since only a single type of road or water vehicle is allowed in a particular instance. Thus, each node is able to have a single value for the number of loads/unloads that can be performed, rather than an amount for each vehicle type. Since the maximum number of loads/unloads at a terminal node only pertain to particular arcs, we define the set

$$V_i^{\overrightarrow{T}} = \{j \mid (i,j) \in A^T \setminus A^S \wedge \eta_{ij} \neq \perp\}$$

to be the set of nodes connected by outgoing, non-stationary, non-LT arcs from the time-space terminal node i, and

$$V_i^{\overleftarrow{T}} = \{j \mid (j,i) \in A^T \setminus A^S \wedge \eta_{ji} \neq \perp\}$$

to be the set of nodes from incoming, non-stationary, non-LT arcs from the time-space terminal node i.

7.3.2 Demand

We consider ITT demands using a multi-commodity flow. ITT requires a multi-commodity flow in order to adequately model the flow of containers between terminals at various times. Let Θ be the number of demands, and $o_\theta \in V$, $d_\theta \in V$, $a_\theta \in \mathbb{Z}^+$, $r_\theta \in \{0,\ldots,\tau-1\}$, and $u_\theta \in \{0,\ldots,\tau-1\}$ be the origin node, destination node, amount of containers, release period and due period of demand $1 \leq \theta \leq \Theta$. Each demand is also associated with a penalty function $p_\theta : \{0,\ldots,\tau-1\} \to \mathbb{R}$ that describes the penalty to be assessed based on the delivery time of each container, where p_θ is 0 if the container is delivered early or on time. That is, $p_\theta(t) = 0$ for all $t \leq u_\theta$. We use a piecewise linear function to model the lateness, but any function can be used, even in the IP, because of the discretization of time in the model.

Our demand structure does not differentiate between different types of containers, such as refrigerated, out-of-gauge or dangerous goods. While some types of containers may require special handling procedures for ITT, the vast majority of containers do not, and we therefore treat all containers equally for the purposes of this model. For modeling purposes, let

$$V_\theta^D = \{i \in V^T | \lfloor i/\tau \rfloor \neq o_\theta \wedge \lfloor i/\tau \rfloor \neq d_\theta\}$$

be the set of time-space nodes that do not match either the origin or destination of θ in the base graph.

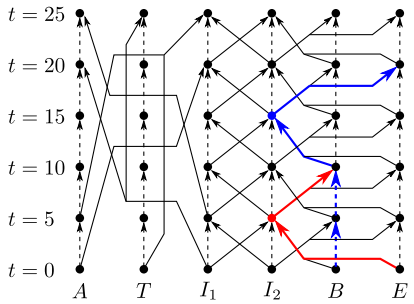


Figure 7.2: A time-space graph for an ALV in the port of Hamburg over a 25 minute period with a 5 minute discretization. Solid arcs represent roads and dashed arcs represent stationary arcs. The terminals A, T, B and E are connected through intersections I_1 and I_2. A vehicle path from E to B, passing through intersection node I_2, is highlighted in red, and a demand path from B to E (release at time 0) in blue.

7.3.3 Time-space Graph Example

Figure 7.2 shows the time-space graph for our model of the port of Hamburg over a 25 minute period with a 5 minute discretization. The majority of the connections are on public roadways, especially between B, E, I_1 and I_2. Each arc defines the travel time required for a vehicle based on its connections, and the travel time required can be varied at different times in the model to model rush hour or significant traffic disruptions. Note that we do not show arcs destined for nodes later than time 25, for reasons of clarity. In the graph, there is a single vehicle located at node E at $t = 0$. A demand originates at B at $t = 0$ and must be brought to E before time 15 to avoid a penalty of 5 units per discretized period. The red line shows the path of the ALV as it drives with no containers from E to B. The ALV loads the container at $t = 10$ when it arrives at B and leaves in the same time period, as ALVs load containers quickly, and returns to E. The blue line shows the location of the demand from the time it is released until it is delivered. First, the demand stays at B on stationary arcs for two time periods before it is picked up by the ALV and transported to E through I_2. Since the container is delivered one time period late, a penalty of 5 units is incurred. Note that this is the optimal penalty for this instance, as delivering the container earlier is not possible due to the location of the ALV at time 0.

7.3.4 IP Model

Using our time-space graph, we now present an IP model to solve ITT problems that minimizes the late delivery of containers. Our IP model differentiates itself from other time-space models in the way it handles LT arcs and nodes, as well as the parallel flows of vehicles and containers. The goal of the model is to minimize the penalty incurred from delivering containers past their due date.

Parameters

The following parameters are used in our IP model.

n	Number of nodes in the base graph.
τ	Number of time periods.
V	Set of nodes in the base graph.
V^T	Set of nodes in the time-space graph.
A^T	Set of arcs in the time-space graph.
A^{LT}	Set of LT arcs.
Θ	Number of demands.
$In(i)$	Set of nodes with an arc to node $i \in V^T$.
$Out(i)$	Set of nodes with an arc from node $i \in V^T$.
V_θ^D	Set of nodes excluding any time-space node that matches the origin or destination of θ.
$V_i^{\overrightarrow{T}}$	Outgoing, non-stationary, non-LT arcs from node $i \in V^T$.
$V_i^{\overleftarrow{T}}$	Incoming, non-stationary, non-LT arcs from node $i \in V^T$.
o_θ	Origin node in V of demand θ.
d_θ	Destination node in V of demand θ.
a_θ	Amount of containers in demand θ.
r_θ	Release time step of demand θ.
u_θ	Due time step of demand θ.
p_θ	Late delivery penalty function.
$\delta_{ij\theta}$	Equal to 0 iff arc $(i,j) \in A^T$ is a stationary arc from the demand origin of θ or is an LT arc.
c_{ij}	Maximum number of vehicles on arc $(i,j) \in A^T$.
m_i	Maximum number of container load/unload moves during a time period at node i.
m_i^{LT}	Maximum number of LT vehicle load/unload moves during a time period at node i.
s_i	Amount of vehicles present at node $i \in V^T$ at the start of optimization.
γ_i	Maximum vehicle throughput of node $i \in V^T$.
μ_{ij}	Maximum container capacity of a vehicle on arc $(i,j) \in A^T$.

Variables

We introduce two sets of decision variables to control the flow of containers through the model. Let $x_{ij} \in \{0, \ldots, c_{ij}\}$ be the amount of vehicles on arc $(i,j) \in A^T \setminus A^{LT}$. We restrict x_{ij} to not include LT arcs, since LT nodes are only a modeling artifact, and therefore require no vehicles to function. Let $y_{ij\theta} \in \{0, \ldots, a_\theta\}$ be the amount of containers flowing on arc $(i,j) \in A^T$ for demand θ.

Objective and Constraints

$$\min \sum_{1\le\theta\le\Theta} \sum_{u_\theta<t<\tau} \sum_{i\in In(d_\theta)} p_\theta(t-u_\theta) y_{id_\theta\theta} \tag{7.1}$$

$$\text{s.t.} \sum_{1\le\theta\le\Theta} \delta_{ij\theta} y_{ij\theta} \le \mu_{ij} x_{ij} \qquad \forall\,(i,j)\in A^T \tag{7.2}$$

$$\sum_{j\in Out(i)} x_{ij} - \sum_{k\in In(i)} x_{ki} \le s_i \qquad \forall\, i\in V^T \tag{7.3}$$

$$\sum_{j\in Out(i)} x_{ij} + \sum_{j\in In(i)} x_{ji} \le \gamma_i \qquad \forall\, i\in V^T \tag{7.4}$$

$$\sum_{j\in Out(o')} y_{o'j\theta} = a_\theta \qquad \forall\, 1\le\theta\le\Theta, o' = \tau(o_\theta-1)+r_\theta+1 \tag{7.5}$$

$$\sum_{j\in Out(i)} y_{ij\theta} - \sum_{k\in In(i)} y_{ki\theta} = 0 \qquad \forall\, 1\le\theta\le\Theta, i\in V_\theta^D \tag{7.6}$$

$$\sum_{\tau d_\theta\le j<\tau(d_\theta+1)} \sum_{i\in In(j)} y_{ij\theta} = a_\theta \qquad \forall\, 1\le\theta\le\Theta \tag{7.7}$$

$$\sum_{1\le\theta\le\Theta} \left(\sum_{j\in V_i^{\vec{T}}} y_{ij\theta} + \sum_{j\in V_i^{\overleftarrow{T}}} y_{ji\theta} \right) \le m_i \qquad \forall\, 0\le i\le n\tau \tag{7.8}$$

$$\sum_{1\le\theta\le\Theta} (y_{ij\theta} + y_{ji\theta}) \le m_i^{LT} \qquad \forall\, 1\le i\le n\tau, j = n\tau+i \tag{7.9}$$

The objective (7.1) is to minimize the lateness of container delivery, in which each container delivered late is penalized as a function of the lateness. That is, each arc entering a time-space node of a demand destination is assigned a cost computed from the penalty function $p_\theta(t-u_\theta)$, where θ is the demand, t is the time step, and u_θ is the due time of the demand.

The amount of containers flowing on an arc is constrained in (7.2) to be no more than the total capacity of all the vehicles traveling on an arc, where $\delta_{ij\theta}\in\{0,1\}$ is set to 0 iff *i)* $\lfloor i/\tau \rfloor = o_\theta$ and $(i,j)\in A^S$, or *ii)* $(i,j)\in A^{LT}$. Therefore, $\delta_{ij\theta}$ takes the value 0 only when (i,j) is a stationary arc from the origin of demand θ or (i,j) is an LT arc. When a container is present on a stationary or LT arc, it corresponds to the container not physically moving anywhere. Thus, no vehicle is required to carry the container.

Constraints (7.3) are vehicle flow balance constraints that allow vehicles to flow within the time-space graph. The constraints enforce that there are never more vehicles leaving a node than the number of vehicles entering a node added to the number of vehicles that start at the node. Note that this constraint allowed vehicles to travel empty anywhere in the graph. Constraints (7.4) cap the number of vehicles entering and leaving a node in a particular time step. This cap on vehicle throughput is primarily used on intersections. However, it can also be used on nodes to model a gate entry/departure queue.

Constraints (7.5), (7.6) and (7.7) flow the containers through the network. Constraints (7.5) bind the origin of a demand to have an outflow of the amount of containers in the demand, (7.6) ensures an exact container flow balance through the graph, and (7.7) requires that all of the containers of a demand arrive at the destination node, respectively. Note that containers are allowed to arrive early at their destination node, and may then flow on the stationary arcs until their due time is reached.

The time it takes to load and unload containers is taken into account in constraints (7.8), which bound the maximum number of load/unload moves that can be made at a node in a single time step for each vehicle type. Note that this constraint does not control the flow of containers over stationary arcs or LT arcs, in order to allow containers to remain at their origin or to be loaded on to LT vehicles. The containers flowing between LT nodes and the terminals they represent are handled in constraints (7.9). By allowing only m_i^{LT} containers between time-space node i and its LT node, barges must be (un)loaded at the rate the quay cranes can handle.

There are several small improvements we can make to the model to help the presolver reduce the problem size. First, we set all $y_{ij\theta}$ variables to 0 if $r_\theta > (i-1) \bmod \tau$, i.e., the release time of θ is greater than the starting time of node i. Second, we set all $y_{ij\theta}$ variables to 0 if $j = o_\theta$, meaning demands are not allowed to re-enter their origin node. While this seems like an obvious observation, this prevents the IP from searching through many solutions in which the paths of demands travel in loops.

7.3.5 Flow-first Solution Approach

In addition to simply solving our IP model from the previous subsection in an IP solver in one go, we propose a two-step solution method called the *flow-first* approach. First, we post constraints (7.5) through (7.9), which correspond to the container multi-commodity flow, and solve the corresponding IP. This IP generally takes little time to solve, as solvers like CPLEX have strong cuts for multi-commodity flow structures. We then post constraints (7.2), (7.3) and (7.4), which control the flow of vehicles and ensure that containers are carried by vehicles. Finally, we re-solve the IP using the solution to the pure multi-commodity flow problem as a starting solution (albeit infeasible).

Although the multi-commodity flow solution is essentially always infeasible due to the routing of vehicles, CPLEX is able to use the solution as an internal guide for tackling the larger problem. The idea behind this is that many of the paths of the containers in the multi-commodity flow will still be valid once vehicles are brought into the problem, and by computing these paths without the complication of vehicles, CPLEX can focus on completing the transportation of more difficult containers when solving the full problem. Alternatively, CPLEX can use a repair mechanism to try to fix the solution. We determine the effectiveness of this approach in the following computational evaluation.

7.4 Computational Evaluation

The following section describes the evaluation of our IP model on generated datasets based on the port of Hamburg and the Maasvlakte 1 & 2 area of the port of Rotterdam. We show that our model gives valuable and actionable information for the planning of ITT systems.

7.4.1 Data Generation

We generated an artificial dataset based on communications from the port of Rotterdam regarding the Maasvlakte 2 expansion and data gathered on the internet for the port of Hamburg. We use the same vehicle properties as in [41] for AGVs, ALVs and MTSs in terms of load/unload times, vehicle velocity and capacity, giving AGVs, ALVs and MTSs velocities of 5.0 m/s, 4.0 m/s, 6.6 m/s, respectively. AGVs and ALVs may carry a single container each, and an MTS can carry up to 5 containers. We allow AGVs to load and unload containers at a rate of 30 moves per hour per crane available. We allow MTSs a move rate of 35 moves per hour, corresponding to the efficiency gained by quickly loading multiple containers in a set of trailers. Note that we do not take a detailed view of MTSs involving the coupling and decoupling of tractor units. We allow ALVs an essentially infinite load and unload rate. ALVs are reported in [41] to require about a minute to load or unload a container, but unlike AGVs and MTSs, they do not have to form a queue and wait for a crane. Since our model is unable to take into account such fine grained interactions between ALVs, we allow them a fast load/unload time at ports. We distribute vehicles amongst terminals uniformly at time zero, and randomly distribute remaining vehicles if the number of vehicles is not divisible by the number of terminals.

We assume barges in the model have a capacity of 50 containers and travel at a rate of 2.2 m/s (slightly under 5 knots), as this is a common maximum speed in harbors. Barges can load and unload containers at a rate of 30 moves per hour, and we assume two cranes are used to load/unload barges, giving 60 moves per hour.

Congestion We take a view of congestion in which roadways (arcs) and intersections (nodes) have a maximum throughput per time period. While such a model lacks the detail of the directions vehicles turn and does not handle specific vehicle to vehicle interactions, it is able to provide a reasonably good restriction on port throughput, which is what is most important for our model. Furthermore, a detailed view of congestion in which vehicle movements are precisely modeled would require significantly more variables, likely too many to find a solution when several hundred vehicles are present.

Demands Each demand is generated by choosing two different terminals within a port uniformly at random, then choosing an amount of containers less than 50 containers that must be transferred between the two terminals, and finally setting a release time

b	Type	h	$\|A^T\|$	$\|V^T\|$
0	U	AGV	1510	576
		ALV	1506	
		MTS	1512	
	RH	AGV	1506	
		ALV	1502	
		MTS	1508	
2	U	AGV	3726	960
		ALV	3722	
		MTS	3728	
	RH	AGV	3722	
		ALV	3718	
		MTS	3724	

Table 7.1: The number of nodes ($|V^T|$) and arcs ($|A^T|$) in the time-space graph for the Hamburg instances.

and due time. In order to prevent obviously infeasible instances, we choose the release time uniformly at random in the range $[0, t^{max} - 2time(a, b)]$, where t^{max} is the maximum time of the model minus a constant t^c, which we set to 1 hour for Hamburg instances and 2 hours for Maasvlakte instances, a and b are the two terminals chosen for the demand, and $time(a, b)$ is the minimum time required for the slowest vehicle type to travel between a and b. We choose the values of t^c based on the size of the port areas in order to prevent instances that will be infeasible nearly regardless of the type of vehicle that is used to solve an instance. We compute $time(a, b)$ using a simple all-pairs shortest path algorithm. We multiply this time by a factor of 2 to further prevent clearly infeasible instances. If $t^{max} - 2time(a, b)$ is a negative number, we choose a release time in the interval $[0, t^{max}/10]$. We then choose a due time of the demand that is between the release time plus $time(a, b)$, with the maximum value being t^{max}. Note that this could mean some demands are infeasible for delivery for slow vehicles. We view this as a necessary part of the evaluation of such vehicle types, as only generating data that is feasible for all vehicle types could make slow vehicles look just as effective as fast vehicles at delivering demands under tight deadlines, which is not always the case.

Each demand is associated with a penalty function to discourage lateness. We use three penalty functions for demands representing low, medium and high priority containers. We use a triangular distribution to assign the majority of the demands to be low priority demands, roughly 30% to be medium priority, and the rest to be high priority (slightly over 10%).

7.4.2 Hamburg

We model the port of Hamburg with 4 terminals and 2 intersections in the base graph, and compute ITT performance over a period of 8 hours with a 5 minute discretization. We generate 10 sets of demands for 500, 1000, 1500, and 2000 containers. We then generate instances with varying numbers of road vehicles (50, 100, 150, and 200), with 2 barges and with no barges (labeled by b in the following tables), as well as with

uniform traffic (U) and rush hour traffic (RH). We model rush hour traffic on specific arcs that contain non-port traffic. During the first and last hour of a rush hour instance, these arcs take longer to traverse than during the other 6 hours.

Table 7.1 shows the size of the time space graph for the Hamburg instances. The size of the graph is independent of the number of containers. The number of arcs differs between vehicle types because of their different speeds. Some connections at the end of the 8 hour period are not possible to complete with slow vehicles. The number of arcs for RH instances is slightly lower for similar reasons. That is, some arcs that take longer to traverse due to rush hour, and therefore do not have end points within the 8 hour window.

We solved the instances in our Hamburg dataset to optimality using CPLEX 12.4 with AMD Opteron 2344 HE processors with a maximum of 3 GB of RAM per process and a maximum CPU time of one hour. Table 7.2 gives the average penalty, in thousands, for the Hamburg instances, which we present to show the kind of data our model can provide to decision makers. The penalty is averaged across all 10 runs of each combination of number of containers, number of barges, infrastructure type, and vehicle type. In cases where the model was unable to find an optimal penalty, we use the LP relaxation. We are able to to do this because the LP relaxation value is very close to the optimal solution value across our dataset. Out of the 382 instances with 500 containers with and without barges that we solved to optimality, 91% of them had an LP relaxation value at the root node equal to the optimal solution. This percentage also holds in the case of 1000 and 1500 container instances. We were unable to solve many 2000 container instances to optimality, but of the 7 that we did solve, all of them had an LP relaxation equal to the optimal solution. While this could be a case of survivorship bias, we note that the problems solved range in CPU time from just a few seconds to almost an entire hour. When we lower the timeout to half an hour instead of a full hour, we see little difference in the percentage of solved instances with their LP relaxation equal to the optimal solution, even though less instances have been solved.

In computing the average penalty, we also combine the results of running the *all-at-once* model, in which we plug our entire IP model into CPLEX, and the flow-first model from the Section 7.3.5. If one method solves a problem to optimality, that objective is used, or if an approach finds that an instance is infeasible, the instance is declared infeasible even if the other approach timed out attempting to prove this. We provide an analysis of the run time of these two approaches following our discussion of the solutions found by the approaches.

Overall, MTSs offer the lowest penalized delivery across all instances. However, AGVs in combination with barges provide nearly as good performance in the 500 and 1000 container cases, offering only a 6.5% increase in penalty over MTSs and barges on both uniform and rush hour instances. With only 50 vehicles, only MTSs are able to provide delivery for all of the cargo in all instances with 500 containers, and in most of the instances with 1000 containers. Our model shows that, under the given assumptions, 100 road vehicles are generally sufficient for performing ITT, and that adding extra vehicles is unable to provide less penalties, due to road congestion.

$\|C\|$	b	Type	h	50		100		150		200	
				Pen.	Inf.	Pen.	Inf.	Pen.	Inf.	Pen.	Inf.
500	0	U	AGV	24.59	1	8.27	0	8.27	0	8.27	0
			ALV	98.65	4	14.04	0	14.04	0	14.04	0
			MTS	5.96	0	5.96	0	5.96	0	5.96	0
		RH	AGV	40.42	1	8.27	0	8.27	0	8.27	0
			ALV	134.60	4	14.04	0	14.04	0	14.04	0
			MTS	5.96	0	5.96	0	5.96	0	5.96	0
	2	U	AGV	5.39	0	4.98	0	4.98	0	4.98	0
			ALV	9.84	0	9.17	0	9.17	0	9.60	1
			MTS	3.49	0	3.49	0	3.49	0	3.88	1
		RH	AGV	5.39	0	5.05	0	4.98	0	4.98	0
			ALV	10.14	1	9.17	0	9.19	0	9.60	1
			MTS	3.49	0	3.49	0	3.49	0	3.49	0
1000	0	U	AGV	-	10	7.02	0	2.30	1	3.05	0
			ALV	-	10	29.36	2	9.10	0	8.96	0
			MTS	0.93	0	0.93	0	0.93	0	0.93	0
		RH	AGV	-	10	14.90	0	4.52	0	4.48	0
			ALV	-	10	42.35	2	11.18	0	10.70	0
			MTS	2.33	0	2.33	0	2.33	0	2.33	0
	2	U	AGV	0.78	5	0.94	4	1.07	3	0.94	4
			ALV	9.60	2	4.54	4	4.08	4	4.08	4
			MTS	0.60	2	0.48	2	0.43	2	0.43	2
		RH	AGV	2.88	4	2.31	4	2.25	3	2.25	3
			ALV	6.85	3	5.68	4	6.85	3	7.72	3
			MTS	1.43	1	1.44	2	1.44	2	1.44	2
1500	0	U	AGV	-	10	-	10	58.57	9	2.17	8
			ALV	-	10	-	10	204.21	9	47.76	8
			MTS	5.26	0	5.26	0	5.26	0	5.26	0
		RH	AGV	-	10	-	10	84.44	9	11.20	8
			ALV	-	10	-	10	-	10	77.55	9
			MTS	5.26	0	5.26	0	5.85	1	5.26	0
2000	0	U	AGV	-	10	-	10	-	10	-	10
			ALV	-	10	-	10	-	10	-	10
			MTS	6.26	6	6.92	3	4.71	5	1.44	7
		RH	AGV	-	10	-	10	-	10	-	10
			ALV	-	10	-	10	-	10	-	10
			MTS	5.74	1	5.30	0	4.67	3	5.30	0

Table 7.2: The average penalty (Pen.) for the Hamburg instances (in thousands) and the number of infeasible instances (Inf.).

The effect of our congestion model, in which intersections have a maximum throughput per time period, can be seen when the number of AGVs and ALVs is increased. The performance of using 100 AGVs and ALVs instead of 50 generally increases across all instance types and numbers of containers. However, moving from 100 to 150 or more vehicles does not have the same increase in performance, despite the increase in capacity. We conclude that this is due to intersections filling with AGVs and ALVs, causing congestion, an important outcome for port authorities to take note of.

We do not include results for barges for 1500 and 2000 container instances as our model timed out on the instances, due to their size. Above 1500 containers, AGVs and ALVs begin to not be sufficient for satisfying all of the demands, and the instances are proven infeasible. The low carrying capacity of AGVs and ALVs is their main drawback, although their slower speed in comparison to MTS systems does not help, either. The faster loading capabilities of ALVs do not seem to outweigh these drawbacks in these scenarios.

We next present CPU time results for the flow-first method and the all-at-once

$\|C\|$	b	Type	h	50				100				150				200			
				All-at-once		Flow-first		All-at-once		Flow-first		All-at-once		Flow-first		All-at-once		Flow-first	
				CPU	TO	CPU	TO	CPU	TO	CPU	TO	CPU	TO	CPU	TO	CPU	TO	CPU	TO
500	0	U	AGV	228.32	0	67.19	0	140.82	0	40.37	0	87.09	0	38.06	0	70.85	0	40.27	0
			ALV	109.99	0	91.70	0	94.26	0	29.30	0	60.11	0	27.45	0	26.62	0	28.92	0
			MTS	16.88	0	10.29	0	29.04	0	10.67	0	11.56	0	10.96	0	8.69	0	11.47	0
		RH	AGV	167.37	0	85.38	0	97.30	0	39.41	0	155.49	0	38.80	0	42.24	0	32.19	0
			ALV	176.23	0	159.41	0	48.68	0	28.78	0	55.27	0	26.31	0	28.61	0	25.48	0
			MTS	7.78	0	11.42	0	11.05	0	10.51	0	11.73	0	10.96	0	8.26	0	8.80	0
	2	U	AGV	2247.75	8	1870.34	8	1431.29	4	1880.01	3	2307.05	5	1248.89	5	1572.34	6	926.79	5
			ALV	2527.08	8	996.59	9	1268.30	6	1074.24	5	2041.89	6	1696.38	1	1213.50	5	1115.30	6
			MTS	826.13	0	568.49	0	1000.04	2	508.61	1	776.61	0	515.31	0	636.35	1	809.96	0
		RH	AGV	1145.94	9	1784.89	9	1459.25	3	1745.30	5	2200.37	6	1849.84	5	1306.47	6	1855.37	3
			ALV	3485.08	9	2575.76	9	1370.41	6	2131.32	5	1850.23	2	1529.20	3	1034.32	4	1079.74	4
			MTS	537.29	0	709.63	1	580.88	0	729.37	0	349.40	0	756.55	0	421.67	0	775.63	0
1000	0	U	AGV	311.19	0	265.78	0	1782.91	2	680.35	0	1605.07	0	371.65	0	1046.37	0	211.99	0
			ALV	184.20	0	204.24	0	1821.98	0	1534.83	1	676.54	0	271.67	0	779.25	0	382.77	0
			MTS	605.93	0	161.87	0	1064.68	0	186.26	0	1070.92	0	205.12	0	889.05	0	226.60	0
		RH	AGV	262.22	0	200.74	0	1794.92	0	567.61	0	977.78	1	439.21	0	1252.96	0	538.26	0
			ALV	144.40	0	146.43	0	2491.01	2	1675.82	2	857.98	0	250.21	0	704.52	0	209.09	0
			MTS	730.71	0	222.06	0	761.98	2	173.08	0	702.04	0	197.64	0	863.59	0	184.74	0
	2	U	AGV	3547.19	6	-	10	3508.94	6	-	10	3507.11	7	-	10	3539.90	7	-	10
			ALV	3516.36	9	-	10	3535.23	7	-	10	3540.58	6	-	10	3526.48	6	-	10
			MTS	3469.34	9	2185.23	9	2940.14	7	1360.37	9	3538.27	9	1049.30	9	3509.75	7	-	10
		RH	AGV	3533.51	6	-	10	3535.93	7	-	10	3533.78	7	-	10	3548.33	7	-	10
			ALV	3483.57	9	-	10	3541.48	6	-	10	3521.27	7	-	10	3528.98	6	-	10
			MTS	3485.23	9	-	10	3557.39	8	-	10	2635.16	7	2937.56	8	3544.85	8	1312.35	9
1500	0	U	AGV	358.01	0	531.58	0	1028.61	1	1109.88	0	656.48	1	774.13	1	1103.11	1	1008.99	3
			ALV	487.42	0	525.03	0	910.38	0	825.71	0	1154.84	1	807.81	0	957.81	0	901.71	1
			MTS	2751.50	6	2233.11	7	1829.49	4	719.57	0	2708.66	5	1278.82	5	1662.53	3	1017.96	0
		RH	AGV	485.14	0	362.92	0	1028.61	1	1139.64	0	496.51	1	839.46	1	1076.15	1	739.97	1
			ALV	505.89	0	523.73	0	435.24	1	726.22	0	608.54	2	1069.39	1	967.77	0	706.70	0
			MTS	2071.16	8	2236.47	6	1997.59	3	863.15	3	2074.72	3	745.84	2	1911.20	3	828.75	3
2000	0	U	AGV	771.55	0	677.82	0	1060.56	1	1003.28	0	1703.92	1	1199.20	0	1055.61	0	1579.64	0
			ALV	700.93	0	437.04	0	557.33	0	887.45	0	978.51	0	750.23	0	666.92	1	1610.26	0
			MTS	-	10	3575.37	8	-	10	2564.89	9	-	10	3554.95	8	2356.41	8	617.60	9
		RH	AGV	752.73	0	1164.78	0	730.81	0	516.18	2	867.92	2	1027.32	2	596.03	1	1273.61	2
			ALV	714.33	0	1199.34	0	478.02	0	812.61	1	486.44	1	1050.15	0	680.19	0	1191.28	0
			MTS	-	10	-	10	3553.14	9	2803.71	9	-	10	3117.29	7	-	10	606.06	9

Table 7.3: The CPU times in seconds and number of timeouts (TO) for the all-at-once and flow-first approaches on the Hamburg instances.

approach for solving the IP model. The flow-first method's first step, in which a multi-commodity flow problem is solved, generally only takes several seconds in CPLEX. Even with side constraints like our loading and unloading restrictions and LT arc capacity constraints (Constraints (7.8) and (7.9), respectively), the resulting model poses no large difficulties.

Table 7.3 shows the CPU times and number of timeouts on the Hamburg instances for both the all-at-once and flow-first approaches. In terms of CPU time, the flow-first method outperforms the all-at-once approach on non-barge instances of 1000 containers or less. On 500 container instances, the average CPU time of the flow-first method is 53% of the all-at-once approach non-barge instances. On barge instances, however, the average performance is nearly identical, with the flow-first approach slightly outperforming the all-at-once method on AGV and ALV instances, but underperforming on MTS instances. For 1000 container instances, the flow-first method has an average CPU time of 380.56 seconds as opposed to 956.11 for the all-at-once approach, meaning it requires only slightly under 40% of the CPU time of the all-at-once approach, ignoring

timeouts. Despite the flow-first method's numerous timeouts on 1000 container barge instances, the all-at-once approach is so close to the timeout for most instances that the average times are not greatly different between the two approaches.

On 1500 container instances, the flow-first approach has an average CPU time (including timeouts at 3600 seconds) on 1500 container instances of only 1242.51 seconds, as opposed to 1539.91 seconds of the all-at-once approach, a savings of 19%. However, on 2000 container instances, flow-first only achieves an average run time of 1810.73 seconds (including timeouts as before) against 1921.41 seconds for the flow-first method, an increase of roughly 6%.

In terms of timeouts, the all-at-once approach achieves fewer timeouts than the flow-first approach across the entire dataset, with only 116 timeouts versus 126 for 50 vehicles, 98 versus 105 timeouts for 100 vehicles, and 91 versus 105 timeouts for 200 vehicles. In the case of 150 vehicles, the flow first method achieves 2 fewer timeouts than all-at-once, with only 98 timeouts as opposed to 100. However, a majority of these timeouts are due to barge instances, which the flow-first method is unable to deal with once the number of containers is greater than 500. Looking only at non-barge instances, the flow-first method has only 113 timeouts as opposed to 136 for all-at-once. We hypothesize that the loading and unloading restrictions of barges using LT arcs makes finding a good multi-commodity flow solution in the flow-first approach more difficult than in the road transportation only case, and in future work we will investigate adding cuts to prevent this from happening.

We conclude that the flow-first method is best suited to non-barge instances, even though it has competitive performance on barge instances in terms of CPU time. However, given that the two approaches often have large differences in performance even on the same instance, a heuristic selection approach, such as the ISAC method [74], could be employed to choose whether or not to apply the flow-first method to instances in future work.

7.4.3 Maasvlakte 1 & 2

Our Maasvlakte 1 & 2 dataset is based off of instances generated with 10 sets of demands in a similar fashion as the Hamburg instances over a 10 hour time period with an 8 minute discretization. We have increased the time period due to the larger size of the Maasvlakte area, which contains 8 terminals distributed across an area of roughly 15 km^2. We model the road transportation connections of these terminals using 4 intersections and the waterway connections with three waterway intersections for the 6 terminals with quays. We generated 10 barges for waterway instances because the Maasvlakte area is significantly larger than the port of Hamburg and has more terminals. We set the maximum due time of demands to all be at most two hours before the end of the time period, although deliveries may still occur in the last two hours. We solve these instances on AMD 6386SE processors with a maximum of 3GB of RAM per process using CPLEX 12.5. We solve all Maasvlakte instances with the flow-first method.

b	Type	h	$\|A^T\|$	$\|V^T\|$
0	U	AGV	2806	900
		ALV	2804	
		MTS	2812	
	RH	AGV	2798	
		ALV	2796	
		MTS	2808	
	FC	AGV	3102	
		ALV	3100	
		MTS	3108	
10	U	AGV	5802	1575
		ALV	5800	
		MTS	5808	
	RH	AGV	5794	
		ALV	5792	
		MTS	5804	
	FC	AGV	6098	
		ALV	6096	
		MTS	6104	

Table 7.4: The number of nodes and arcs in the time-space graph for the Maasvlakte 1 & 2 instances.

In addition to the uniform traffic and rush-hour instances we solved for the port of Hamburg, we propose an infrastructure addition in the Maasvlakte area to show that our model can, with little modification of the underlying graph, model novel infrastructure components. *Fast-connector* instances model two dedicated tunnels (or bridges) that connect a terminal to a key intersection, and then travels further to another intersection. The fast connector significantly shortens the distance vehicles must travel to reach the far ends of the Maasvlakte area. This modification to our instances is realized simply by adding extra arcs to the base graph.

Table 7.4 shows the size of the time-space graph for the Maasvlakte 1 & 2 instances, where b is the number of barges, the instance type is either uniform traffic (U), rush-hour (RH) or fast-connector (FC), and h is the non-barge vehicle type. Instances without barges have 900 nodes, and instances with barges 1575. As in the case of the Hamburg instances, the number of arcs varies slightly between vehicle types due to their differing speeds.

Table 7.5 displays the average solution penalty (in thousands) across the 10 instances solved, along with the number of instances found to be infeasible/the number of timeouts. As in the presentation of the Hamburg instances, we compute the average solution penalty using the optimal objective for all instances that are solved to optimality and the value of the LP relaxation for instances that have timed out. The LP relaxation turns out to be a rather tight bound on the optimal solution found in most cases. In fact, across the 1037 instances for which we have an optimal solution, the value of the LP relaxation matches the solution value on 1020 of them, a total of 98.3%. These instances are not trivial to solve, either, with many taking over 1000 seconds to find a solution.

We solve instances with 500, 1000, 1500 containers with and without barges, as well

as instances with 2000 containers without barges. Our instances with barges proved to be too large for CPLEX, indicating that further work is needed to scale to very large ports with large numbers of containers. Nonetheless, our model is capable of reaching real-world container throughput volumes for actual ports. Several elements in the table have no penalty value computed, and instead are labeled with a dash. In these few cases, the LP relaxation could be solved on the problems that were not proven infeasible.

The lowest penalty across all amounts of cargo tends to be achieved by the MTS systems. Despite their slow loading capabilities, they have two key advantages over AGVs and ALVs. First, they can carry up to 5 containers, meaning less MTS vehicles are needed to service large amounts of demands. Second, they travel faster than AGVs or ALVs, allowing them to cover the large distances of the Maasvlakte area more effectively. Without barges, MTSs have only 28% of the penalty of AGV and ALV instances with 500 containers and 100 vehicles in uniform and rush hour scenarios, and 28% and 15% of the penalty of AGV and ALV instances, respectively, for 1500 container scenarios with 200 vehicles. Barges alleviate some of the stress on AGV and ALV systems, reducing the gap between AGVs/ALVs and MTSs by 36% and 38%, respectively, for 500 container instances with 100 vehicles, and by 87% and 85% for 1500 container instances with 200 vehicles. Barges seem to have little effect on MTSs since they already have the carrying capacity to handle high container volumes. Note that these results do not indicate that ports should always use MTSs. Rather, these results give ports an indication of the delivery delay they will experience by using one option over another.

The gap in performance between AGVs/ALVs and MTSs is narrowed in the FC instances, as the distances between different parts of the port become more manageable. In the case of 500 containers with 200 vehicles or more, the fast connector closes the penalty gap by 50%, bringing the penalty of AGVs and ALVs down to only 3.75 away from the MTS penalty. With 2000 containers, the fast connector reduces the number of infeasible instances for AGVs with all three amounts of vehicles. Port authorities can use this information to weigh the cost of such infrastructure changes with the cost savings they can possibly achieve with automated as opposed to manned systems.

The effect of our congestion model, in which intersections have a maximum throughput per time period, can be seen when the number of AGVs and ALVs is increased. The performance of using 200 AGVs and ALVs instead of 100 generally increases across all instance types and numbers of containers. However, moving from 200 to 300 vehicles does not have the same increase in performance, despite the increase in capacity. We conclude that this is due to intersections filling with AGVs and ALVs, an important outcome for port authorities to take note of.

Rush hour (RH) instances show a marked increase in penalty over uniform traffic instances for AGVs and ALVs, but not MTSs, in instances with 1000 containers or more. This provides evidence that MTSs are a better solution than AGVs or ALVs when port and non-port traffic are mixed, as the high numbers of AGVs and ALVs necessary to service cargo contribute to the rush hour traffic.

The fast connector (FC) instances tend to show little improvement as the number

$\|C\|$	b	Type	h	100			200			300		
				Pen.	Inf.	TO	Pen.	Inf.	TO	Pen.	Inf.	TO
500	0	U	AGV	17.19	0	0	12.53	0	0	12.53	0	0
			ALV	17.71	0	0	12.61	0	0	12.61	0	0
			MTS	4.93	0	0	4.93	0	0	4.93	0	0
		RH	AGV	17.19	0	0	12.53	0	0	12.53	0	0
			ALV	17.71	0	0	12.61	0	0	12.61	0	0
			MTS	4.93	0	0	4.93	0	0	4.93	0	0
		FC	AGV	8.71	0	0	8.71	0	0	8.71	0	0
			ALV	8.71	0	0	8.71	0	0	8.71	0	0
			MTS	4.96	0	0	4.96	0	0	4.96	0	0
	10	U	AGV	12.81	0	6	12.01	0	4	12.01	0	7
			ALV	12.89	0	3	12.09	0	5	12.09	0	4
			MTS	4.93	0	0	4.93	0	0	4.93	0	0
		RH	AGV	12.81	0	5	12.01	0	5	12.01	0	7
			ALV	12.89	0	2	12.09	0	5	12.09	0	2
			MTS	4.93	0	0	4.93	0	0	4.93	0	0
		FC	AGV	8.71	0	2	8.71	0	1	8.71	0	0
			ALV	8.71	0	1	8.71	0	1	8.71	0	2
			MTS	4.96	0	0	4.96	0	0	4.96	0	0
1000	0	U	AGV	23.09	1	2	29.21	0	0	29.21	0	0
			ALV	45.03	1	3	32.07	1	0	31.52	0	0
			MTS	4.74	0	0	4.74	0	0	4.74	0	0
		RH	AGV	36.91	1	2	46.31	0	0	38.78	0	0
			ALV	23.59	1	5	42.55	1	0	41.38	0	0
			MTS	4.82	0	0	4.82	0	0	4.82	0	0
		FC	AGV	12.15	0	1	12.15	0	3	12.15	0	2
			ALV	12.15	0	0	12.15	0	0	12.15	0	0
			MTS	5.89	0	0	5.89	0	0	5.89	0	0
	10	U	AGV	19.70	0	10	14.44	0	10	25.56	0	10
			ALV	27.22	0	10	27.06	0	10	27.06	0	10
			MTS	4.57	0	3	4.57	0	3	4.57	0	3
		RH	AGV	29.53	0	10	29.53	0	10	29.53	0	10
			ALV	30.51	0	10	31.03	0	10	28.18	0	10
			MTS	4.57	0	2	4.57	0	1	4.57	0	2
		FC	AGV	10.66	0	10	11.25	0	10	11.80	0	9
			ALV	12.15	0	10	12.15	0	10	12.15	0	10
			MTS	5.89	0	0	5.89	0	0	5.89	0	0
1500	0	U	AGV	320.96	1	9	20.13	1	7	29.63	1	6
			ALV	-	2	8	39.91	2	2	36.56	2	3
			MTS	5.68	0	0	5.68	0	0	5.68	0	4
		RH	AGV	-	4	6	31.59	1	2	24.07	0	5
			ALV	-	4	6	38.56	1	2	49.38	3	0
			MTS	5.68	0	2	5.68	0	2	5.68	0	0
		FC	AGV	19.42	0	4	15.88	0	1	15.88	0	1
			ALV	18.38	0	6	16.86	0	0	10.38	0	3
			MTS	6.14	0	2	6.14	0	3	6.14	0	6
	10	U	AGV	9.67	0	10	7.34	0	10	13.33	0	10
			ALV	11.69	0	10	10.69	0	10	13.54	0	10
			MTS	5.46	0	9	5.46	0	6	5.46	0	8
		RH	AGV	9.67	0	9	10.56	0	9	6.38	0	9
			ALV	8.84	0	9	16.99	0	9	20.08	0	9
			MTS	5.46	0	6	5.46	0	6	5.46	0	6
		FC	AGV	8.26	0	10	8.52	0	10	10.29	0	10
			ALV	10.50	0	10	11.11	0	10	8.77	0	10
			MTS	6.14	0	1	6.14	0	0	6.14	0	1
2000	0	U	AGV	-	1	9	-	3	7	-	1	9
			ALV	-	4	6	-	5	5	-	5	5
			MTS	9.05	0	6	9.05	0	7	9.05	0	7
		RH	AGV	-	3	6	-	2	7	3.69	1	8
			ALV	-	2	7	-	3	6	-	4	5
			MTS	9.05	0	2	9.05	0	6	9.05	0	7
		FC	AGV	37.85	0	10	19.98	0	8	13.81	0	9
			ALV	-	0	10	26.43	0	7	26.43	0	8
			MTS	10.84	0	6	10.84	0	9	10.84	0	10

Table 7.5: Average delivery penalty (Pen.), the number of infeasible instances (Inf.) and timeouts (TO) for the Maasvlakte 1 & 2 areas.

$\|C\|$	b	Type	h	100				200				300			
				Cols	Rows	NZ	CPU	Cols	Rows	NZ	CPU	Cols	Rows	NZ	CPU
500	0	U	AGV	12	31	129	234.30	13	34	134	292.33	13	33	131	296.77
			ALV	12	31	128	200.14	13	33	136	167.77	13	33	134	149.72
			MTS	13	34	126	72.15	13	34	123	65.13	13	34	122	59.86
		RH	AGV	12	31	127	235.73	13	33	133	265.63	13	33	129	242.45
			ALV	12	30	126	176.63	12	33	134	147.74	12	33	132	145.28
			MTS	13	34	125	60.96	13	34	122	66.36	13	34	121	61.28
		FC	AGV	13	37	157	314.28	13	37	152	352.48	13	37	148	328.96
			ALV	13	37	157	182.81	13	37	153	165.08	13	37	152	172.67
			MTS	13	37	144	77.88	13	37	135	81.01	13	37	135	73.46
	10	U	AGV	24	70	266	853.38	24	70	261	367.48	24	70	257	417.90
			ALV	24	70	265	738.92	24	70	262	1211.84	24	70	261	1101.01
			MTS	24	70	249	240.89	24	70	246	259.45	24	70	246	257.72
		RH	AGV	24	70	264	883.99	24	70	259	1133.96	24	70	255	1097.66
			ALV	24	69	263	1623.02	24	70	261	781.92	24	70	260	718.49
			MTS	24	70	248	244.26	24	70	244	259.48	24	70	244	260.35
		FC	AGV	24	73	281	820.97	24	73	275	393.49	24	73	271	590.02
			ALV	24	73	280	673.04	24	73	277	396.45	24	73	275	653.05
			MTS	24	73	265	276.14	24	73	256	269.80	24	73	256	280.26
1000	0	U	AGV	18	52	221	1643.55	22	67	281	1376.75	22	67	276	1534.97
			ALV	18	52	217	1315.84	22	67	280	1144.40	22	67	276	1173.18
			MTS	22	68	275	305.75	22	68	246	343.82	22	68	244	327.41
		RH	AGV	17	50	212	1362.70	22	66	278	1419.55	22	66	272	1418.00
			ALV	18	54	227	999.90	22	66	276	794.49	22	66	272	867.48
			MTS	22	67	272	315.31	22	67	244	318.25	22	67	242	397.22
		FC	AGV	23	74	322	1238.28	23	74	317	1811.16	22	74	311	1784.32
			ALV	22	74	320	1400.91	22	74	316	1111.14	22	74	311	1595.10
			MTS	23	75	310	354.08	23	75	283	470.17	22	75	269	569.03
	10	U	AGV	42	139	538	-	42	140	534	-	42	140	528	-
			ALV	42	139	536	-	42	140	533	-	42	139	528	-
			MTS	42	140	522	998.52	42	140	491	1053.41	42	140	489	1071.63
		RH	AGV	42	139	535	-	42	139	532	-	42	139	525	-
			ALV	42	138	533	-	42	139	530	-	42	139	525	-
			MTS	41	139	518	994.31	41	138	487	1047.40	41	138	484	920.15
		FC	AGV	42	145	569	-	42	146	564	-	42	146	558	1728.74
			ALV	42	145	568	-	42	146	564	-	42	146	559	-
			MTS	42	146	555	1112.12	42	146	528	1089.37	42	146	512	1166.89
1500	0	U	AGV	30	96	410	3372.53	30	96	408	2286.73	30	96	404	2359.99
			ALV	30	95	409	1726.32	30	95	406	2157.89	30	95	403	2174.19
			MTS	31	97	402	668.46	31	97	374	1072.48	30	97	348	1463.10
		RH	AGV	30	94	404	2657.88	30	94	401	2253.62	30	94	397	2259.15
			ALV	30	94	402	2402.91	30	94	400	1953.54	30	94	397	2279.97
			MTS	30	96	398	1029.04	30	96	370	1053.60	30	96	345	1135.80
		FC	AGV	30	102	443	2609.38	31	106	459	2147.40	31	106	455	2454.49
			ALV	31	106	461	2489.79	31	106	459	2081.42	31	106	455	2117.69
			MTS	31	107	453	839.60	31	107	424	1385.94	31	107	400	1879.76
	10	U	AGV	56	196	759	-	57	200	770	-	57	200	766	-
			ALV	56	196	757	-	57	199	769	-	57	199	766	-
			MTS	57	200	758	3192.33	57	200	727	2490.89	57	200	699	1991.30
		RH	AGV	56	195	752	-	57	199	766	-	57	199	762	-
			ALV	56	194	750	-	57	199	765	-	57	198	762	-
			MTS	57	198	751	2453.92	57	198	720	2179.10	56	198	693	2538.90
		FC	AGV	57	206	809	-	57	208	814	-	57	208	810	-
			ALV	57	206	808	-	57	208	814	-	57	208	810	-
			MTS	57	209	805	2567.69	57	209	775	2617.02	57	209	750	2571.67
2000	0	U	AGV	39	127	546	3026.41	39	127	544	1983.65	39	127	541	2647.58
			ALV	39	127	543	2935.70	39	127	542	3024.40	39	126	540	2252.41
			MTS	38	125	526	1346.18	39	129	516	1901.32	39	129	488	1839.17
		RH	AGV	39	125	537	3235.71	39	125	536	2610.28	38	125	533	1368.74
			ALV	38	125	535	2947.57	38	125	534	3201.56	38	125	531	1595.43
			MTS	39	128	538	1531.82	39	128	511	1857.69	39	128	483	2271.66
		FC	AGV	39	141	613	-	39	141	612	2940.82	39	141	609	3417.00
			ALV	39	140	612	-	39	140	610	3386.42	39	140	608	3556.95
			MTS	40	142	610	2071.77	40	142	582	1465.52	39	142	554	-

Table 7.6: Size of the Maasvlakte 1 & 2 instances in terms of the number of post-processed IP columns, rows and non-zeros (NZ) in thousands, and the average CPU times in seconds for solving all instances that did not timeout.

of vehicles is increased in both the 500 and 1000 container cases without barges, and in the 500 container case with barges. We suspect that this is due to congestion in port intersections. Without the fast connector, vehicles spend more time traveling on arcs and are unable to crowd the intersections. However, with the fast connector, more cargo can be delivered on time, and this means more crowding of the intersections. This could provide important information to a port in determining how many vehicles they will need given an infrastructure change like in the FC instances. Since less vehicles are required, the port can offset the costs of the infrastructure development. Furthermore, this shows that our model can be used to test out scenarios to find out where the new bottlenecks in a port will be. In our instances, the port would benefit from increasing intersection throughput in addition to building the fast connector.

In our model, MTSs are the only way of servicing large amounts of containers within the 10 hour time period; when the number of containers reaches 2000, AGV and ALV instances begin timing out or returning infeasible solutions. In fact, on the ALV and AGV instances with 2000 containers, an LP relaxation is often difficult to compute within the one hour time limit, indicating there probably is no solution at all.

We report the number of columns, rows and non-zero entries (in thousands) in the post-processed IP in CPLEX 12.5, along with the average CPU times in seconds required to solve an instance in Table 7.6. Although the underlying model is the same for various numbers of vehicles, with the only difference being the domain of the x_{ij} variables, the number of post-processed rows, columns and non-zeros can vary, but not significantly. The solution time does not have any strong correlation with the number of vehicles in the model. However, the inclusion of barges always makes finding a solution harder, due to the increased problem size. MTS instances are overall easier than AGV and ALV instances, requiring only 24% and 41% of the CPU time of AGV and ALV instances for 500 containers without barges, respectively, despite the fact that the models are essentially the same size. For 1500 container instances without barges, MTS instances solve 47% and 54% faster for AGVs and ALVs, respectively.

As the container volumes grow, the solution times approach, and often exceed, one hour of CPU time. Developing faster algorithms to solve ITT problems is therefore valuable future work. However, we do not view this as a major hindrance to the utilization of our model in practice, as the planning of ports and port expansions is a long term process for which significant computation power can be obtained.

7.5 Chapter Summary

Inter-terminal transportation (ITT) is a key factor in the decision making process for the construction of new ports and expansion of existing ports. These projects represent critical, long-term and expensive infrastructure investments that require effective analysis of ITT for decision makers. To this end, we presented an integer programming model to minimize container delivery delay that takes into account the key components of ITT, including traffic congestion, multiple vehicle types and loading/unloading times, and

arbitrary terminal configurations. We also provided a two stage approach for solving the model to optimality.

Our model scales to the sizes of real-world ports, time periods, and container throughput, as shown using examples from the Maasvlakte area of the port of Rotterdam and the port of Hamburg, and provides important analysis on not only the feasibility, but also the delay of containers reaching their destinations. Our model of ITT is the first to incorporate optimization of vehicle routes and container flows in order to provide ports and terminals with the best performance a particular configuration of vehicles and infrastructure is capable of delivering. The model represents a particularly difficult class of time-space models, in which interacting vehicle flows, a multi-commodity flow, and congestion constraints all interact.

For future work, we intend to use our model to help analyze the costs of building new infrastructure and/or purchasing new types of vehicles versus the improvements in port efficiency and reductions in delays. In addition, we will use this model to guide the input into a discrete event simulation of ports, thus providing a complete view of the impact of strategic decisions on port efficiency. Our model could also be used to assist ports in determining what vehicle characteristics would best fit their port from an engineering perspective. Finally, we also intend to investigate the impact of long distances on port design, such as those found in the port of Shanghai, China, or the port of Rotterdam outside of the Maasvlakte area.

Chapter 8

Container Stowage Planning Complexity

Containers are constructed such that they can be stored space efficiently directly on top of each other in stacks. Containers are stored this way both in stationary storage areas, such as depots and port terminal yards, and moving storage areas such as bays of container ships. Another characteristic of these storage areas is that containers arrive and depart at discrete points in time. For a typical depot and yard, trucks load and unload containers daily, while containers stowed in bays of ships are loaded and unloaded at different ports. This, and the fact that stacks only can be accessed from the top, complicates the decision of where to stow containers in a storage area. We call this general discrete optimization problem *stowage planning*. Stowage planning is hard because containers to be retrieved from the storage area must be at or near the top positions of stacks.

If a container must be removed from a stack in order to retrieve a container underneath it, that container is said to be *overstowing* the container being retrieved. The process of removing an overstowing container is called *shifting*. Thus, the goal of stowage planning is to minimize shifting or, equivalently, to minimize overstowage. This is particularly important in the bays of container ships, because shifting requires that the overstowing container is moved from the ship to the terminal yard and back, which is very expensive.

Despite the practical importance of container stowage problems, the complexity of several key stowage related problems remains unknown. To this end, we first solve an open problem stated in [10] by showing that a change from uncapacitated to capacitated stacks in their k-shift problem reduces its complexity from NP-complete to polynomial. We do this by providing an algorithm that for any choice of the number of stacks and stack capacities solve the problem in polynomial time. Although our algorithm is impractical, the fact that such a problem can be solved in polynomial time gives hope that efficient algorithms can be found in other areas within the container stowage domain. This chapter is based on our paper [151].

8.1 The Capacitated k-Shift Problem

The Capacitated k-Shift Problem (CkSP) is a decision problem that asks whether a set of containers that must be stored and retrieved at discrete time points can be stowed with less than k shifts in a fixed number of stacks with limited capacity. Formally, an instance of the CkSP is a tuple $\langle n, C, In, Out, S, m, k\rangle$, where n is the number of time points, C is a finite set of containers, $In(c) \in \{1, \ldots, n\}$ ($Out(c) \in \{1, \ldots, n\}$) is the time point that container c must be stowed in (retrieved from) one of the stacks, S is a finite set of stacks, $m \in \mathbb{N}$ is the maximum number of containers that each stack can hold at any time, and k is the maximum number of allowed shifts.

The question is whether the containers can be assigned to the stacks such that at most k shifts are required to retrieve them. Formally, is there an assignment $A : C \to S$ that is within the stack capacity (i.e., $\forall t \in \mathbb{N}, s \in S,\ |\{c \mid A(c) = s, In(c) \leq t < Out(c)\}| \leq m$) that requires at most k shifts (i.e., $|\{w \in C \mid \exists v, A(v) = A(w) \wedge In(v) < In(w) < Out(v) \ \wedge \ In(w) < Out(v) < Out(w)\}| \leq k$)?

8.2 Related Work

Although there is a significant amount of work for solving container stowage problems, (see [113] for a full overview of the relevant approaches), the theoretical complexity of stowage planning problems has not received significant consideration in the literature. The uncapacitated version of the CkSP where $m = \infty$ has been shown to be NP-complete for $|S| \geq 4$ by a reduction from the coloring of overlap graphs [10] and is known to be polynomial for $|S| < 4$ [9, 153]. For $m = 1$, the CkSP is solvable in time polynomial in C using a minimal cost flow formulation for interval scheduling on identical machines [15]. However, even for $m = 2$, it is not known whether the CkSP can be solved with a polynomial time algorithm.

8.3 A Polynomial Time Algorithm

In this section, we show that a polynomial time algorithm exists to solve the CkSP for any choice of $|S|$ and m.

Example 8.1. Consider the CkSP depicted in Figure 8.1, in which $C = \{c_1 \ldots c_{13}\}$, $In(c_1, \ldots, c_4) = 1$, $In(c_5, c_6) = 2$, $In(c_7, \ldots, c_{10}) = 3$, $In(c_{11}) = 4$, $In(c_{12}, c_{13}) = 5$, $Out(c_1, \ldots, c_4) = 3$, $Out(c_5) = 4$, $Out(c_6) = 6$, $Out(c_7, \ldots, c_{10}) = 5$, $Out(c_{11}, c_{12}, c_{13}) = 6$, $S = \{s_1, s_2\}$, $m = 3$. The answer to this CkSP is "yes" for all $k \geq 3$.

We define a *configuration* as a function $q : \{1, \ldots, m\} \times \{1, \ldots, |S|\} \to \{1, \ldots, n\}$ assigning slots to discharge times. In other words, when a container c is loaded into a slot in a particular stack, we must only make note of its discharge time ($Out(c)$) in the configuration, yielding at most n^σ configurations at any time point, where $\sigma = m|S|$ represents the total number of slots for stowing containers. Let $\perp$ represent

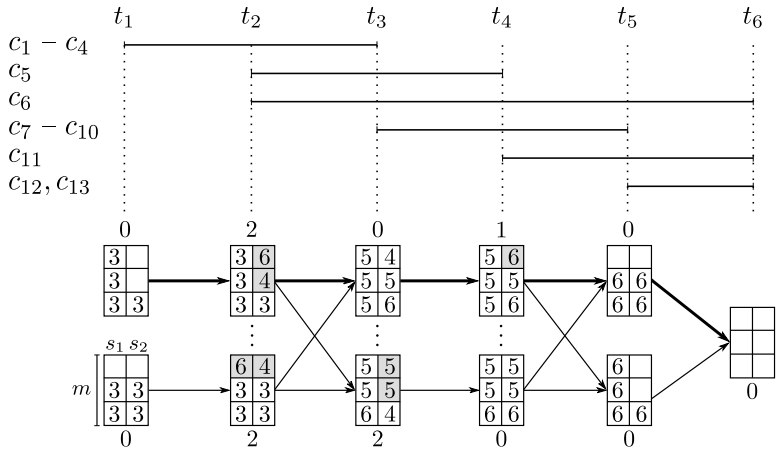


Figure 8.1: A CkSP instance with $m = 3$,$|S| = 2$ and $|C| = 13$. The instance is a "yes"-instance for any $k \geq 3$. Possible configurations are shown at each time step with the discharge time of the container in each slot. Gray slots contain containers that must be shifted, and each configuration is labeled with the number of containers that will be shifted at that timestep. A path through the configurations represents a complete stowage plan, and an optimal plan is shown with bold arcs.

an empty configuration. Given a configuration q and a time point t, let $Q(q,t)$ be the set of configurations that q may *transition* to at time t. We allow a transition between configurations q and q' at time t only when it is possible to re-stow the stacks after unloading containers (including shifted containers) such that each slot in q with a container that has a discharge time later than t and is not overstowing a container being discharged at t, has the same slot assignment in q'. Additionally, $Q(\perp, t)$ provides all of the possible configurations at time t, with $Q(\perp, 1)$ representing the initial (empty) configuration.

We will now give an algorithm that exploits this polynomially bounded number of configurations to solve the CkSP in polynomial time for any choice of m and S. The minimal number of shifts is given by

$$k^* = \min_{q \in Q(\perp,1)} \mathit{CkSP\text{-}DP}(q, 1),$$

$$\mathit{CkSP\text{-}DP}(q,t) = \begin{cases} 0 & \text{if } t = n, \\ \mathit{shifts}(q,t) + \min\limits_{q' \in Q(q,t)} \{\mathit{CkSP\text{-}DP}(q', t+1)\} & \text{otherwise.} \end{cases}$$

The function $\mathit{CkSP\text{-}DP}(q,t)$ represents the minimum number of shifts necessary for unloading containers in configuration q from time t to time n, where $\mathit{shifts}(q,t)$ returns the number of shifts for the current configuration at time t, and $Q(q,t)$ is as previously defined.

Given a configuration q at time t, our dynamic program computes the number of shifts q requires at time t, followed by the minimum cost transition to a state q' at time

$t + 1$. When $t = n$, all containers are discharged and no shifts are required, thus the cost of all states when $t = n$ is 0.

We return a "yes" answer to the CkSP if $k^* \leq k$ and "no" otherwise. Figure 8.1 shows some of the states of the dynamic program as it solves the problem given in Example 8.1. Each state represents a configuration and shows the discharge ports of the containers loaded in each slot. An arc connects configuration q at time t to configuration q' at time $t + 1$ if $q' \in Q(q, t)$, meaning there is a transition between the two configurations. A path through the states represents an assignment of containers to stacks, and the total number of shifts is given by the sum of the number of shifts in each state (shown above and below each state).

We first show that computing *shifts*(q, t) and determining the configurations that can be transitioned to, $Q(q, t)$, both take polynomial time to compute. We then leverage these results to show that computing k^* takes polynomial time, and thus, solving the CkSP takes polynomial time.

Lemma 8.1. *Given a configuration q and a time t, computing $Q(q, t+1)$ has complexity $O(\sigma n^\sigma)$.*

Proof. First, note that there are n^σ possible configurations in total since each slot can be assigned any discharge port. We must determine whether a transition from q exists for each of the possible configurations q' at time $t + 1$. This check involves ensuring that all containers not leaving or entering the stacks at time $t+1$ in q and q' are in the same positions. Note that shifted containers are considered as leaving and re-entering the stacks, meaning they are allowed to change positions between q and q'. Checking configurations q and q' takes time $O(\sigma)$, since each slot in each stack is examined in constant time. We perform this check between q and all n^σ configurations, giving a total time of $O(\sigma n^\sigma)$. □

Theorem 8.1. $k^* = \min_{q \in Q(\perp,1)}$ *CkSP-DP*$(q, 1)$ *can be computed in polynomial time for any choice of m and $|S|$.*

Proof. The minimal number of shifts, k^*, is computed through a dynamic programming procedure which investigates $O(n^{\sigma+1})$ different states, since there are n time steps and n^σ possible configurations at each one. The function *shifts*(q, t) computes the number of overstowages at each state, and is clearly polynomial, as it involves looking at each stack in q at time t and counting the number of shifts required. Thus, since *shifts*(q, t) takes polynomial time, and by Lemma 8.1, processing each state takes polynomial time for a fixed m and S. □

Lemma 8.2. *CkSP-DP*(q, t) *returns the minimal number of shifts for any configuration q and time t.*

Proof. We use a proof by induction on n. In the base case, when $t = n$,

$$\textit{CkSP-DP}(q, n) = 0, \forall q \in Q(\perp, n),$$

since all containers are unloaded at time n. For the inductive step, assume that all configurations at time t have been assigned the minimal number of shifts, i.e. *CkSP-DP*(q,t) returns the minimum number of shifts for any $q \in Q(\perp,t)$, and the dynamic program must compute *CkSP-DP*$(q',t-1)$ for all $q' \in Q(\perp,t-1)$. The minimal number of shifts for *CkSP-DP*$(q',t-1)$ is therefore

$$\mathit{shifts}(q',t-1) + \min_{q \in Q(q',t-1)} \{\mathit{CkSP\text{-}DP}(q,t)\},$$

because in order to unload containers in state q' at time $t-1$, $\mathit{shifts}(q',t-1)$ shifts must be performed, and the latter part of the term holds by the inductive hypothesis. □

Theorem 8.2. $k^* = \min_{q \in Q(\perp,1)} \mathit{CkSP\text{-}DP}(q,1)$ *is the minimum number of shifts.*

Proof. By Lemma 8.2, *CkSP-DP*(q,t) returns the minimal number of shifts for any q and t. Since k^* takes the minimum number of shifts computed by *CkSP-DP*(q,t) on each of the initial configurations, k^* must equal the minimum number of shifts. □

8.4 Chapter Summary

We investigated the complexity of stowage planning problems. We showed that the capacitated k-shift problem (CkSP) is solvable in polynomial time for any choice of stacks and stack capacities, which is an open problem from [10]. Even though the algorithm we propose is not practical, knowing that the CkSP is solvable in polynomial time under certain conditions is an interesting result for the scientific community, and we hope that this will spur research into devising faster algorithms for solving k shift problems that can be put into practice. The complexity of several variations of the CkSP remain unknown, such as when the stack capacity is variable and the number of stacks is either fixed or is variable, but greater than the number of time points, or when the stack capacity is fixed and the number of stacks is variable and greater than three.

Chapter 9

Conclusion

This dissertation investigates a central problem of the liner shipping industry, the liner shipping fleet repositioning problem (LSFRP), and solves it using a plethora of techniques both ignoring and taking into account container flows. We provide the first problem description and mathematical models of the LSFRP in the literature.

The central question of this dissertation is to determine whether an algorithm can be developed to create real-world realizable liner shipping fleet repositioning plans within a reasonable amount of CPU time. We answer this question with a clear "yes", and present algorithms for two versions of the LSFRP that are able to find optimal or near optimal solutions in only minutes of CPU time.

For the LSFRP without cargo flows (NCLSFRP), we compare four different approaches, including automated planning, mixed-integer programming, constraint programming and the novel linear temporal optimization planning method. We show that constraint programming has the best performance of all the approaches, but the least extensible model. The NCLSFRP is one of a number of problems with time-dependent task costs that are receiving an increased focus in the literature [83].

We also introduce three mathematical models of the LSFRP with cargo flows using a specialized graph structure to model problem specific constraints. We evaluate these models, which we call the arc flow, path based, and node flow models, on a real world dataset from our industrial collaborator. We show that nearly all of the dataset can be solved to optimality within several hours, with many problems solvable in only a few seconds. We also introduce two heuristic procedures based on simulated annealing and late acceptance hill climbing in order to provide high quality solutions to problem instances that we cannot solve to optimality. We also compare our approaches against a reference solution from our industrial collaborator and show that we can achieve nearly $14 million of additional profit compared to their repositioning plan.

The LSFRP is a new optimization problem, and there is much future work to be done to bring the algorithms provided in this dissertation into the industry. In particular, the algorithms and models in this dissertation could be brought into an interactive decision support tool to assist repositioning coordinators in creating repositioning plans. On the algorithmic side of the LSFRP, more work is needed in order to solve large problems

with non-fixed visitation times to optimality without running out of memory. There are a number of model additions to the LSFRP that could increase the problem's realism and its applicability in industry, such as being able to send vessels to receive repairs, or handle more detailed cabotage restrictions.

There are a number of opportunities for integrating the LSFRP into other areas of liner shipping research. For example, combining the LSFRP with fleet deployment would create a fleet re-deployment problem that would try to move vessels to places in the network where they could be more profitable, but balance this with the costs of repositioning the vessel. Additionally, the LSFRP can be combined with network design to create a network transition problem (similar to that discussed in [5]), in which a liner shipping network is transformed into a new one that can better carry customer demands while taking into account the costs of transitioning.

Secondary problems

This dissertation also addresses two other liner shipping topics: inter-terminal transportation (ITT) and a theoretical problem in container stowage.

The secondary question regards ITT, and asks whether a general model of ITT can be developed to minimize delay that handles arbitrary types of material handling equipment and transportation infrastructure, and can be solved to optimality within an hour of CPU time. We answer this question with "yes" and present a novel mathematical model of ITT. We introduce a two step procedure for solving ITT problems to optimality. We evaluate our approach on data based on the ports of Hamburg, Germany and Rotterdam, Netherlands. We show that port authorities can gain significant insight into how to structure the transportation infrastructure of their port and what vehicles they should purchase in order to minimize the delay of containers reaching their destination terminal.

In the area of ITT, a number of notable problems still remain for future work. Determining the optimal mix of vehicles for minimizing delay could help ports lower their costs, but the current model only supports a single vessel type on land and one on rails or in the harbor. Future ITT models should support multiple vehicle types interacting within a transportation network. Furthermore, ports could optimize the specific features they desire of vehicles, such as their speed and carrying capacity, in order to custom design vehicles fitting a specific port area.

In the area of container stowage, we solve an open problem called the capacitated k-shift problem with fixed stacks and stack heights by presenting a polynomial time algorithm for the problem. A number of theoretical questions remain in the area of container stowage, such as when only stack heights or the number of stacks are not fixed.

Outlook

Over the coming decades, ports and shipping lines must overcome numerous challenges in order to keep up with ever increasing containerized trade volumes. With container volumes set to exceed 170 million TEU in 2013 [154] and forecasts showing further growth [155], the importance of effective algorithms for decision support at sea and on land grows as well. This dissertation has addressed several pressing issues in the container shipping domain, bringing to light new problems and providing answers for others. More collaboration is needed with the liner shipping industry to explore the many combinatorial challenges of containerized trade. With ever more cargo being shipped with containers, addressing these challenges is critical for increasing the affordability of goods and improving the environmental sustainability of global trade.

Bibliography

[1] A. Abuhamdah. Experimental result of late acceptance randomized descent algorithm for solving course timetabling problems. *International Journal of Computer Science and Network Security*, 10:192–200, 2010.

[2] T. Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[3] R. Agarwal and Ö. Ergun. Ship scheduling and network design for cargo routing in liner shipping. *Transportation Science*, 42(2):175–196, 2008.

[4] J.F. Álvarez. Joint routing and deployment of a fleet of container vessels. *Maritime Economics and Logistics*, 11(2):186–208, June 2009.

[5] M.W. Andersen. *Service Network Design and Management in Liner Container Shipping Applications*. PhD thesis, Technical University of Denmark, Department of Transport, 2010.

[6] P. Angeloudis and M.G.H. Bell. A review of container terminal simulation models. *Maritime Policy & Management*, 38(5):523–540, 2011.

[7] C. Ansotegui, M. Sellmann, and K. Tierney. A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms. In I.P. Gent, editor, *Principles and Practice of Constraint Programming (CP-09)*, volume 5732 of *Lecture Notes in Computer Science*, pages 142–157. Springer, 2009.

[8] C. Archetti and M.G. Speranza. Vehicle routing problems with split deliveries. *International Transactions in Operational Research*, 19(1-2):3–22, Jan 2012.

[9] A. Aslidis. Minimizing of overstowage in containership operations. *Operations Research*, 90:457–471, 1990.

[10] M. Avriel, M. Penn, and N. Shpirer. Container ship stowage problem: Complexity and connection to the coloring of circle graphs. *Discrete Applied Mathematics*, 103:271–279, 2000.

[11] J Benton, A.I. Coles, and A.J. Coles. Temporal planning with preferences and time-dependent continuous costs. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS-12)*, 2012.

[12] C. Bessiere. Constraint propagation. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 29 – 83. Elsevier, 2006.

[13] C. Bierwirth and F. Meisel. A survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research*, 202(3):615–627, May 2010.

[14] A.L. Blum and M.L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.

[15] K.I. Bouzina and H. Emmons. Interval scheduling on identical machines. *Journal of Global Optimization*, 9:379–393, 1996.

[16] D. Briskorn and S. Hartmann. Simulating dispatching strategies for automated container terminals. *Operations Research Proceedings 2005*, pages 97–102, 2006.

[17] B.D. Brouer, J.F. Alvarez, C.E.M. Plum, D. Pisinger, and M.M. Sigurd. A base integer programming model and benchmark suite for liner shipping network design. *Transportation Science*, 2012.

[18] B.D. Brouer, J. Dirksen, D. Pisinger, C.E.M Plum, and B. Vaaben. The Vessel Schedule Recovery Problem (VSRP) – A MIP model for handling disruptions in liner shipping. *European Journal of Operational Research*, 224(2):362–374, 2013.

[19] E.K. Burke and Y. Bykov. A late acceptance strategy in hill-climbing for exam timetabling problems. In *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT-08)*, 2008.

[20] E.K. Burke and Y. Bykov. The late acceptance hill-climbing heuristic. Technical Report CSM-192, University of Stirling, 2012.

[21] M. Caserta, S. Schwarze, and S. Voß. A mathematical formulation and complexity considerations for the blocks relocation problem. *European Journal of Operational Research*, 219(1):96–104, May 2012.

[22] M. Christiansen. Decomposition of a combined inventory and time constrained ship routing problem. *Transportation Science*, 33(1):3–16, 1999.

[23] M. Christiansen, K. Fagerholt, B. Nygreen, and D. Ronen. Maritime transportation. *Handbooks in operations research and management science*, 14:189–284, 2007.

[24] M. Christiansen, K. Fagerholt, B. Nygreen, and D. Ronen. Ship routing and scheduling in the new millennium. *European Journal of Operational Research*, 228(3):467–483, Aug 2013.

[25] M. Christiansen, K. Fagerholt, and D. Ronen. Ship routing and scheduling: Status and perspectives. *Transportation Science*, 38(1):1–18, 2004.

[26] J. Clausen, A. Larsen, J. Larsen, and N.J. Rezanova. Disruption management in the airline industry–concepts, models and methods. *Computers & Operations Research*, 37(5):809–821, 2010.

[27] A.I. Coles, A.J. Coles, A.G. Olaya, S. Jiménez, C.L. López, S. Sanner, and S. Yoon. A survey of the seventh international planning competition. *AI Magazine*, 33(1):83–88, 2012.

[28] A.I. Coles, M. Fox, K. Halsey, D. Long, and A. Smith. Managing concurrency in temporal planning using planner-scheduler interaction. *Artificial Intelligence*, 173(1):1–44, 2009.

[29] A.I. Coles, M. Fox, D. Long, and A. Smith. Planning with problems requiring temporal coordination. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*, 2008.

[30] A.J. Coles, A.I. Coles, A. Clark, and S.T. Gilmore. Cost-sensitive concurrent planning under duration uncertainty for service level agreements. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS-11)*, June 2011.

[31] A.J. Coles, A.I. Coles, M. Fox, and D. Long. Temporal planning in domains with linear processes. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*, 2009.

[32] A.J. Coles, A.I. Coles, M. Fox, and D. Long. Temporal planning in domains with linear processes. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*, July 2009.

[33] A.J. Coles, A.I. Coles, M. Fox, and D. Long. Forward-chaining partial-order planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS-10)*, May 2010.

[34] A.J. Coles, A.I. Coles, M. Fox, and D. Long. COLIN: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research*, 44:1–96, 2012.

[35] B.J. Cudahy. *Box Boats: How Container Ships Changed the World*. Fordham University Press, 2006.

[36] W. Cushing, S. Kambhampati, Mausam, and D. Weld. When is temporal planning *really* temporal planning? In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 1852–1859, 2007.

[37] J.R. Daduna, R. Stahlbock, and S. Voß. Systems for linking seaport container terminals and dedicated satellite terminals. Working paper, presented at The 12th International Conference on Logistics and Maritime Systems (LOGMS), Aug 22–24, 2012, University of Bremen, Germany, 2012.

[38] A. Delgado, R.M. Jensen, K. Janstrup, T.H. Rose, and K.H. Andersen. A constraint programming model for fast optimal stowage of container vessel bays. *European Journal of Operational Research*, 220(1):251–261, 2012.

[39] J. Desrosiers and M.E. Lübbecke. A primer in column generation. *Column Generation*, pages 1–32, 2005.

[40] M.B. Do and S. Kambhampati. Sapa: A multi-objective metric temporal planner. *Journal of Artificial Intelligence Research*, 20(1):155–194, 2003.

[41] M.B. Duinkerken, R. Dekker, S.T.G.L. Kurstjens, J.A. Ottjes, and N.P. Dellaert. Comparing transportation systems for inter-terminal transport at the Maasvlakte container terminals. *OR Spectrum*, 28(4):469–493, 2006.

[42] Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54(1):7–22, 1991.

[43] S. Edelkamp and J. Hoffmann. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report No. 195, Institut für Informatik, 2003.

[44] K. Fagerholt, G. Laporte, and I. Norstad. Reducing fuel emissions by optimizing speed on shipping routes. *Journal of the Operational Research Society*, 61(3):523–529, 2009.

[45] T. Feydy and P.J. Stuckey. Lazy clause generation reengineered. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP-09)*, volume 5732 of *Lecture Notes in Computer Science*, pages 352–366, 2009.

[46] R.E. Fikes and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[47] M. Fox and D. Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[48] M. Fox and D. Long. Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research*, 27:235–297, 2006.

[49] J. Frank, M.A.K. Gross, and E. Kürklü. Sofia's choice: an ai approach to scheduling airborne astronomy observations. In *Proceedings of the 16th conference on Innovative Applications of Artifical Intelligence (IAAI-04)*, pages 828–835. AAAI Press, 2004.

[50] A. Gerevini and A. Saetti. Temporal planning with problems requiring concurrency through action graphs and local search. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS-10)*, 2010.

[51] A. Gerevini, A. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs. In *Journal of Artificial Intelligence Research*, volume 28, pages 239–290, 2003,.

[52] A.E. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5):619668, 2009.

[53] M. Ghallab and H. Laruelle. Representation and control in IxTeT, a temporal planner. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, volume 94, pages 61–67, 1994.

[54] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[55] P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859, 1961.

[56] Google. Google OR-Tools. http://code.google.com/p/or-tools/, 2012.

[57] M. Grunow, H.O. Günther, and M. Lehmann. Strategies for dispatching AGVs at automated seaport container terminals. In K.H. Kim and H.O. Günther, editors, *Container Terminals and Cargo Systems*, pages 155–178. Springer Berlin Heidelberg, 2007.

[58] Gurobi Optimization. Gurobi optimizer reference manual. *URL: http://www.gurobi.com*, 2012.

[59] P. Haslum, B. Bonet, and H. Geffner. New Admissible Heuristics for Domain-Independent Planning. In *Proceedings of the 20th AAAI Conference on Artificial Intelligence (AAAI-05)*, 2005.

[60] P. Haslum, A. Botea, M. Helmert, B. Bonet, and S. Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*, 2007.

[61] P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS-00)*, pages 140–149, 2000.

[62] M. Helmert. Decidability and undecidability results for planning with numerical state variables. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS-02)*, pages 44–53, 2002.

[63] M. Helmert, M. Do, and I. Refanidis. The sixth international planning competition, deterministic track, 2008.

[64] M. Helmert and C. Domshlak. Landmarks, critical paths and abstractions: Whats the difference anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS-09)*, 2009.

[65] M. Helmert, P. Haslum, and J. Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS-07)*, 2007.

[66] J. Hoffmann. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *Journal of Artificial Intelligence Research*, 20:291–341, 2003.

[67] J. Hoffmann and B. Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[68] H.H. Hoos and T. Stützle. *Stochastic local search: Foundations & applications*. Morgan Kaufmann, 2004.

[69] IBM. IBM CPLEX Reference manual and user manual. V12.4, 2012.

[70] A. Imai, K. Shintani, and S. Papadimitriou. Multi-port vs. Hub-and-Spoke port calls by containerships. *Transportation Research Part E: Logistics and Transportation Review*, 45(5):740–757, 2009.

[71] ISO/IEC. Information technology – Programming languages – C++, Third Edition. ISO/IEC 14882:2011, International Organization for Standardization / International Electrotechnical Commission, Geneva, Switzerland, 2011.

[72] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning. *Operations Research*, 37(6):865–892, 1989.

[73] R. Jorgensen. Slow steaming – The full story. `http://www.maersk.com/Innovation/WorkingWithInnovation/Documents/Slow%20Steaming%20-%20the%20full%20story.pdf`. A.P. Moller-Maersk Group. Accessed: 27/3/2013.

[74] S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney. ISAC – Instance-Specific Algorithm Configuration. In H. Coelho, R. Studer, and M. Wooldridge, editors, *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI-10)*, volume 215 of *Frontiers in Intelligence and Applications*, pages 751–756, 2010.

[75] B. Kallehauge, J. Larsen, O.B.G. Madsen, and M.M. Solomon. Vehicle routing problem with time windows. In G. Desaulniers, J. Desrosiers, and M.M. Solomon, editors, *Column Generation*, pages 67–98. Springer, 2005.

[76] D. Karger, C. Stein, and J. Wein. Scheduling algorithms. *CRC Handbook of Computer Science*, 1997.

[77] R.M. Karp. Reducibility among Combinatorial Problems. *Complexity of Computer Computations*, 1972.

[78] M. Katz and C. Domshlak. Implicit abstraction heuristics. In *Journal of Artificial Intelligence Research*, volume 39, pages 51–126, 2010,.

[79] D.E. Kaufman, J. Nonis, and R.L. Smith. A mixed integer linear programming model for dynamic route guidance. *Transportation Research Part B: Methodological*, 32(6):431–440, 1998.

[80] H. Kautz and J.P. Walser. State-space planning by integer optimization. In *Proceedings of the National Conference on Artificial Intelligence*, pages 526–533, 1999.

[81] E. Kelareva, S. Brand, P. Kilby, S. Thiébaux, and M. Wallace. CP and MIP methods for ship scheduling with time-varying draft. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS-12)*, pages 110–118, 2012.

[82] E. Kelareva, P. Kilby, S. Thiébaux, and M. Wallace. Ship scheduling with time-varying draft. In *5th International Workshop on Freight Transportation and Logistics (ODYSSEUS'12)*, 2012.

[83] E. Kelareva, K. Tierney, and P. Kilby. CP Methods for Scheduling and Routing with Time-Dependent Task Costs. In C. Gomes and M. Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 7874 of *Lecture Notes in Computer Science*, pages 111–127. Springer Berlin Heidelberg, 2013.

[84] L.G. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[85] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[86] P. Kissmann and S. Edelkamp. Solving fully-observable non-deterministic planning problems via translation into a general game. In *KI 2009: Advances in Artificial Intelligence*, page 18. Springer, 2009.

[87] N. Kohl, A. Larsen, J. Larsen, A. Ross, and S. Tiourine. Airline disruption management – Perspectives, experiences and outlook. *Journal of Air Transport Management*, 13(3):149–162, 2007.

[88] E. Köhler, K. Langkau, and M. Skutella. Time-expanded graphs for flow-dependent transit times. In R. Möhring and R. Raman, editors, *Algorithms — ESA 2002*, volume 2461 of *Lecture Notes in Computer Science*, pages 599–611. Springer, 2002.

[89] J.E. Korsvik, K. Fagerholt, and G. Laporte. A large neighbourhood search heuristic for ship routing and scheduling with split loads. *Computers & Operations Research*, 38(2):474 – 483, 2011.

[90] E. Kozan. Optimising container transfers at multimodal terminals. *Mathematical and Computer Modelling*, 31(10):235–243, 2000.

[91] H.W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[92] J. Kvarnström. Planning for loosely coupled agents using partial order forward-chaining. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS-11)*, 2011.

[93] Y. Lee and N.Y. Hsu. An optimization model for the container pre-marshalling problem. *Computers & operations research*, 34(11):32953313, 2007.

[94] M. Levinson. *The box: how the shipping container made the world smaller and the world economy bigger*. Princeton University Press, 2006.

[95] H.X. Li and B.C. Williams. Generative planning for hybrid systems based on flow tubes. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS-08)*, 2008.

[96] D. Long and M. Fox. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20:159, 2003.

[97] H. Lourenço, O. Martin, and T. Stützle. Iterated Local Search. *Handbook of Metaheuristics*, pages 320–353, 2003.

[98] H. Marchand, A. Martin, R. Weismantel, and L. Wolsey. Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics*, 123(1):397–446, 2002.

[99] D. McDermott. A heuristic estimator for means-ends analysis in planning. In *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pages 142–149, 1996.

[100] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL – the planning domain definition language. Technical report, 1998.

[101] J. Meyer, R. Stahlbock, and S. Voß. Slow steaming in container shipping. In *45th Hawaii International Conference on System Science (HICSS), 2012*, pages 1306–1314. IEEE, 2012.

[102] N. Muscettola. HSTS: Integrating planning and scheduling. In M. Zweben and M. Fox, editors, *Intelligent Scheduling*, pages 169–212. Morgan Kaufmann, 1993.

[103] G.L. Nemhauser and L.A. Wolsey. *Integer and combinatorial optimization*, volume 18. Wiley New York, 1999.

[104] N. Nethercote, K. Marriott, R. Rafeh, M. Wallace, and M.G. de la Banda. Specification of Zinc and MiniZinc, November 2010.

[105] N. Nethercote, P.J. Stuckey, R. Becket, S. Brand, G.J. Duck, and G. Tack. MiniZinc: Towards a standard CP modelling language. In C. Bessière, editor, *Principles and Practice of Constraint Programming (CP-07)*, volume 4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer, 2007.

[106] V.D. Nguyen and K.H. Kim. A dispatching method for automated lifting vehicles in automated port container terminals. *Computers & Industrial Engineering*, 56(3):1002–1020, 2009.

[107] X. Nguyen and S. Kambhampati. Reviving partial order planning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 459–464, 2001.

[108] T. E. Notteboom and B. Vernimmen. The effect of high fuel costs on liner service configuration in container shipping. *Journal of Transport Geography*, 17(5):325–337, 2009.

[109] T.E. Notteboom. A carrier's perspective on container network configuration at sea and on land. *Journal of International Logistics and Trade*, 1(2):65–87, 2004.

[110] J.A. Ottjes, M.B. Duinkerken, J.J.M. Evers, and R. Dekker. Robotised inter terminal transport of containers. In *Proc. 8th European Simulation Symposium 1996*, pages 621–625, 1996.

[111] J.A. Ottjes, H.P.M. Veeke, M.B. Duinkerken, J.C. Rijsenbrij, and G. Lodewijks. Simulation of a multiterminal system for container handling. *OR Spectrum*, 28(4):447–468, 2006.

[112] E. Ozcan, Y. Bykov, M. Birben, and E.K. Burke. Examination timetabling using late acceptance hyper-heuristics. In *Evolutionary Computation, 2009. (CEC-09). IEEE Congress on*, pages 997–1004. IEEE, 2009.

[113] D. Pacino. *Fast Generation of Container Vessel Stowage Plans – using mixed integer programming for optimal master planning and constraint based local search for slot planning*. PhD thesis, IT University of Copenhagen, June 2012.

[114] D. Pacino, A. Delgado, R.M. Jensen, and T. Bebbington. Fast generation of near-optimal plans for eco-efficient stowage of large container vessels. In *Computational Logistics*, volume 6971 of *Lecture Notes in Computer Science*, pages 286–301. Springer, 2011.

[115] C.H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Dover Publications, 1998.

[116] Y.M. Park and K.H. Kim. A scheduling method for berth and quay cranes. In *Container Terminals and Automated Transport Systems*, pages 159–181. Springer, 2005.

[117] J.S. Penberthy and D. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the 3rd International Conference on Knowledge Representation and Reasoning*, 1992.

[118] J.S. Penberthy and D.S. Weld. Temporal planning with continuous change. In *Proceedings of the National Conference on Artificial Intelligence*. John Wiley & Sons Ltd, 1995.

[119] Port of Rotterdam. Projectorganisatie Maasvlakte 2. `http://www.maasvlakte2.com/en/`. Accessed: 29/04/2013.

[120] B.J. Powell and A.N. Perakis. Fleet deployment optimization for liner shipping: An integer programming model. *Maritime Policy and Management*, 24(2):183–192, Spring 1997.

[121] A.G. Qureshi, E. Taniguchi, and T. Yamada. An exact solution approach for vehicle routing and scheduling problems with soft time windows. *Transportation Research Part E: Logistics and Transportation Review*, 45(6):960–977, 2009.

[122] L.B. Reinhardt and D. Pisinger. A branch and cut algorithm for the container shipping network design problem. *Flexible Services and Manufacturing Journal*, 24(3):349–374, Jul 2011.

[123] Rickmers Group. Rickmers "Pearl" Class 13,100 TEU Container Vessels. `http://www.rickmers.com/fileadmin/rl/download/pdf/pressreleases/13000Naming/100622_Datasheet_13000.pdf`. Accessed: 27/3/2013.

[124] J.P. Rodrigue, C. Comtois, and B. Slack. *The geography of transport systems*. Routledge, 2009.

[125] D. Ronen. Cargo ships routing and scheduling: Survey of models and problems. *European Journal of Operational Research*, 12(2):119–126, 1983.

[126] D. Ronen. Ship scheduling: The last decade. *European Journal of Operational Research*, 71(3):325–333, 1993.

[127] S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472, 2006.

[128] F. Rossi, P. Van Beek, and T. Walsh. *Handbook of constraint programming*. Elsevier Science, 2006.

[129] S.J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Prentice Hall, 2010.

[130] E. Sandewall and R. Rönnquist. A Representation of Action Structures. In *Proceedings of 5th National Conference on Artificial Intelligence*, pages 89–97, 1986.

[131] M.W.P Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation Science*, 29(1):17–29, 1995.

[132] J. Shin and E. Davis. Processes and continuous change in a SAT-based planner. *Artificial Intelligence*, 166(1-2):194–253, 2005.

[133] M.M. Sigurd, N.L. Ulstein, B. Nygreen, and D.M. Ryan. Ship scheduling with recurring visits and visit separation requirements. In G. Desaulniers, J. Desrosiers, and M.M. Solomon, editors, *Column Generation*, pages 225–245. Springer, 2005.

[134] H. Simonis. Sudoku as a constraint problem. In *CP Workshop on modeling and reformulating Constraint Satisfaction Problems*, pages 13–27, 2005.

[135] D.E. Smith, J. Frank, and A.K. Jónsson. Bridging the gap between planning and scheduling. *The Knowledge Engineering Review*, 15(1):47–83, 2000.

[136] S. Smith. Is scheduling a solved problem? *Multidisciplinary Scheduling: Theory and Applications*, pages 3–17, 2005.

[137] R. Stahlbock and S. Voß. Operations research at container terminals: a literature update. *OR Spectrum*, 30(1):1–52, 2008.

[138] M. Stålhane, H. Andersson, M. Christiansen, J.F. Cordeau, and G. Desaulniers. A branch-price-and-cut method for a ship routing and scheduling problem with split loads. *Computers & Operations Research*, 2012.

[139] D. Steenken, S. Voß, and R. Stahlbock. Container terminal operation and operations research – a classification and literature review. *OR spectrum*, 26(1):3–49, 2004.

[140] B. Suman and P. Kumar. A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the Operational Research Society*, 57(10):1143–1160, Oct 2005.

[141] B.G. Tabachnick and L.S. Fidell. *Using multivariate statistics*. Pearson, 2012.

[142] J. Taheri and A.Y. Zomaya. A simulated annealing approach for mobile location management. *Computer communications*, 30(4):714–730, 2007.

[143] K. Tierney. Late Acceptance Hill Climbing for the Liner Shipping Fleet Repositioning Problem. In *Proceedings of the 14th EU/ME Workshop*, pages 21–27, 2013.

[144] K. Tierney, B. Áskelsdóttir, R.M. Jensen, and D. Pisinger. Solving the liner shipping fleet repositioning problem with cargo flows. Technical Report TR-2013-165, IT University of Copenhagen, January 2013.

[145] K. Tierney, B. Áskelsdóttir, R.M. Jensen, and D. Pisinger. Solving the liner shipping fleet repositioning problem with cargo flows. *Under Revision at Transportation Science*, 2013.

[146] K. Tierney, A.J. Coles, A.I. Coles, and R.M. Jensen. A PDDL Domain of the Liner Shipping Fleet Repositioning Problem. Technical Report TR-2012-152, IT University of Copenhagen, 2012.

[147] K. Tierney, A.J. Coles, A.I. Coles, C. Kroer, A.M Britt, and R.M. Jensen. Automated planning for liner shipping fleet repositioning. In L. McCluskey, B. Williams, J.R. Silva, and B. Bonet, editors, *Proceedings of the 22nd International Conference on Automated Planning and Scheduling*, pages 279–287, 2012.

[148] K. Tierney and R.M. Jensen. Temporal optimization planning for fleet repositioning. In *The 5th International Scheduling and Planning Applications woRKshop (SPARK-11)*, 2011.

[149] K. Tierney and R.M. Jensen. The Liner Shipping Fleet Repositioning Problem with Cargo Flows. In Hao Hu, Xiaoning Shi, Robert Stahlbock, and Stefan Voß, editors, *Computational Logistics*, volume 7555 of *Lecture Notes in Computer Science 7555*, pages 1–16. Springer, 2012.

[150] K. Tierney and R.M. Jensen. A node flow model for the inflexible visitation liner shipping fleet repositioning problem with cargo flows. In *Proceedings of the Third International Confernece on Computational Logistics*, Lecture Notes in Computer Science. Springer, May 2013. To Appear.

[151] K. Tierney, D. Pacino, and R.M. Jensen. On the complexity of container stowage planning problems. Under Revision at the Journal of Discrete Applied Mathematics, Jan 2013.

[152] K. Tierney, S. Voß, and R. Stahlbock. A mathematical model of inter-terminal transportation. *European Journal of Operational Research*, May 2013.

[153] W. Unger. The complexity of coloring circle graphs. In *Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science (STACS-92)*, volume 577 of *Lecture Notes in Computer Science*, pages 389–400. Springer, 1992.

[154] United Nations Conference on Trade and Development (UNCTAD). *Review of maritime transport*. 2012.

[155] United Nations Economic and Social Commission for Asia and the Pacific (UNESCAP). *Regional Shipping and Port Development: Container Traffic Forecast 2007 Update*. 2007.

[156] P. van Beek. Backtracking search algorithms. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 85 – 134. Elsevier, 2006.

[157] M. Van Den Briel, T. Vossen, and S. Kambhampati. Reviving integer programming approaches for AI planning: A branch-and-cut framework. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, pages 310–319, 2005.

[158] W. van Hoeve and I. Katriel. Global constraints. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 169 – 208. Elsevier, 2006.

[159] W.J. van Hoeve. The alldifferent constraint: A survey. http://arxiv.org/abs/cs/0105015, 2001.

[160] J. Verstichel and G.V. Berghe. A late acceptance algorithm for the lock scheduling problem. *Logistik Management*, pages 457–478, 2009.

[161] V. Vidal and H. Geffner. Branching and pruning: An optimal temporal POCL planner based on constraint programming. *Artificial Intelligence*, 170(3):298–335, March 2006.

[162] M. Wallace. G12 – Towards the Separation of Problem Modelling and Problem Solving. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 5547 of *Lecture Notes in Computer Science*, pages 8–10. Springer, 2009.

[163] S. Wang and Q. Meng. Sailing speed optimization for container ships in a liner shipping network. *Transportation Research Part E: Logistics and Transportation Review*, 48(3), 2012.

[164] M. Williamson and S. Hanks. Flaw selection strategies for value-directed planning. In *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, volume 23, page 244, 1996.

[165] I.D. Wilson and P. Roach. Container Stowage Planning: A Methodology for Generating Computerised Solutions. *Journal of the Operational Research Society*, 51(11):248–255, 2000.

[166] H.L.S. Younes and R. Simmons. VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research*, 20(1):405–430, 2003.

Appendix A

No Cargo LSFRP PDDL Domain

We provide a detailed description of our PDDL model, as well as the actual forward and reversed PDDL domains for the NCLSFRP. The matching instance files for the domain are available at:
`http://www.decisionoptimizationlab.dk/lsfrp_pddl`.

A.1 PDDL Model

This section covers all of the components of our PDDL model.

A.1.1 Predicates

We provide an overview of the predicates used in our PDDL model. A predicate can be thought of as a state variable as described in Section 4.1 with only two values: true and false. Our model uses a number of predicates to keep track of the state of the vessel and which activities are available during repositioning. For clarity, we remove the type declaration of parameters used in PDDL, and refer readers to the full domain specification in Appendix A for these details.

Vessel state

The status of a vessel is governed by a number of facts:

`(on-init-service ?vessel)` indicates that the vessel is using its initial service, i.e., it has not yet phased out.

`(vessel-at ?vessel ?port)` specifies the location of a vessel (the port it is located at), and is only valid once the vessel has phased out.

`(can-sail ?vessel)` indicates whether or not the vessel is allowed to sail somewhere, and is used to prevent *sequential sailings*, in which a vessel sails in a sequence rather than simply sailing from the first port of the sequence to the last port of the sequence directly.

`(in-transit ?vessel)` is true when after a vessel has phased-out and before it has phased-in.

`(phased-in ?vessel)` indicates the vessel has phased in, meaning its repositioning is complete.

`(sos-or-equipment-allowed ?vessel)` specifies whether an SOS or a sail equipment activity is allowed, as we disallow vessels from performing multiple SOS opportunities, multiple sail equipment actions, or a combination of the two. This fact is true in the initial state and becomes false as soon as an SOS or sail equipment action is used.

Activity Control

The following predicates control which activities are available for use, and are used to prevent multiple vessels from utilizing the same activities.

`(sailing-allowed ?pfrom ?pto)` is true if sailing from port `?pfrom` to `?pto` is allowed.

`(sos-open ?sos ?pfrom ?pto)` indicates whether the SOS opportunity `?sos` is available from SOS start port `?pfrom` to end port `?pto`. Since SOSs are only available at specific time points, this predicate is used to control when the SOS is used.

`(unused ?sos)` specified whether a particular SOS opportunity has been utilized by a vessel. This predicate helps us ensure that SOS opportunities are only used by a single vessel.

`(equipment-sailing ?pfrom ?pto)` is true if there is a sail equipment opportunity available from port `?pfrom` to `?pto`.

Phase In

The phase-in is a particularly difficult for planning techniques and must be modeled carefully to ensure that the PDDL model is solvable. The challenge comes in creating the "block" structure of the phase-in, which ensures that vessels join the goal service with a weekly temporal spacing.

`(block-phase-in-start)` indicates that no vessel has phased in yet.

`(first-phasein-week-defined)` becomes true once the first vessel has phased in.

`(first-phasein-port ?port)` specifies the phase-in port being used.

`(vessel-may-phase-in ?portpi)` is true only at times when vessels may phase-in to a particular service.

`(phasein-week-open)` indicates that a phase-in may occur (as defined through a process, which we will describe in detail later).

Phase Out

The phase-out predicates focus on ensuring that phase-outs only happen at specific times when phase-outs are allowed.

`(allowed-to-phase-out ?vessel)` indicates that a phase-out is allowed to occur. This is used to ensure that the hotel cost period begins with the phase-out.

`(vessel-may-phase-out ?vessel ?port)` is true only at times when the specified vessel may phase out of its initial service at the given port.

Hotel Cost

The hotel cost requires special modeling in PDDL in the form of an envelope action, which we will investigate in more depth in our discussion of the model actions. We use several predicates in order to control the duration of the hotel envelope. During SOS opportunities, the hotel cost is not applicable. This means that, multiple hotel periods may be necessary for vessels using an SOS, and our predicates handle this case.

`(allowed-to-start-cost-calc ?vessel)` is true initially, and allows the hotel cost counting action for a vessel to start. This predicate is also true after SOS actions are completed.

`(allowed-to-end-cost-calc ?vessel)` indicates that a period in which hotel cost is applicable has come to an end, such as when a vessel begins an SOS action or performs its phase-in.

`(cost-calc-mutex ?vessel)` is true during a hotel cost period for a vessel and prevents multiple hotel cost periods from overlapping.

A.1.2 Functions

PDDL functions can be thought of as real valued variables that are part of the planning state. Planning actions can modify the contents of functions in their effect or require certain variables to take specific values in their precondition. In this model, functions are used for representing times when actions are allowed to take place, action costs, and minimum/maximum action durations.

Phasing In

Three functions are used to encode the timing constraints on phasing in:

`(first-phasein-week)` specifies the week in which the first vessel phase-in occurs.

`(weeks-within)` encodes the number of vessels in the problem, which also determines how many phase-in weeks there are.

`(time-elapsed)` provides the amount of time that has passed since time 0. It is updated at a rate of 1 per unit time by a process [48].

Sailing Cost and Times

Due to the time-dependent task costs of the NCLSFRP, costs and times are intimately linked. The function `(total-cost)` represents the cost of a plan. The hotel cost for each vessel is stored in the variable `(hotel-cost ?vessel)`, which is updated by the hotel cost action.

The cost of sailing is calculated by pre-computing the maximum possible sailing cost that is incurred when the vessel sails at its maximum speed, and when the vessel sails slower, a negative cost factor is deducted from the maximum sailing cost based on how much slower than the maximum speed the vessel is sailing. To implement this, we use the following variables:

`(min-time-to-sail ?vessel ?pfrom ?pto)` is the minimum time required to sail between port `?pfrom` to `?pto` for a given vessel, i.e., the ship sails at full speed.

`(max-time-to-sail ?vessel ?pfrom ?pto)` is the maximum time required to sail between port `?pfrom` to `?pto` for a given vessel, i.e., the ship sails at minimum speed.

`(fixed-sail-cost ?vessel ?pfrom ?pto)` is the maximum operational cost of sailing between two ports for a given vessel. This is paid when sailing at the maximum speed.

`(variable-sail-cost ?vessel ?pfrom ?pto)` is a negative number that denotes the cost of taking an extra hour to complete a given sail action.

Sail with Equipment

For sailing with equipment, the minimum and maximum sailing time are encoded in the same way. However, when sailing with equipment the cost of sailing at the minimum speed is zero, and the speed increases linearly as the vessel sails faster. We use the following pair of functions:

`(fixed-eqp-sail-cost ?vessel ?pfrom ?pto)` is the maximum operational cost of sailing with equipment between two ports for a given vessel. This is paid when sailing at the maximum speed.

`(variable-eqp-sail-cost ?vessel ?pfrom ?pto)` is a negative number that denotes the cost of taking an extra hour to complete a given sail action. This variable is set such that when the sailing speed of the vessel is minimum, the cost of the sail equipment action is nothing.

SOS Opportunities

SOS opportunities do not have any operational costs, nor do they incur a hotel cost over their duration. However, SOS opportunities do have a fixed cost relating to the cost of transshipping cargo between the on-service vessel and the repositioning vessel. Thus, we have the following functions:

```
1 (:process time-is-passing
2   :parameters ()
3   :precondition (can-start-time)
4   :effect (increase (time-elapsed) (* #t 1.0)))
```

Figure A.1: The amount of time that has passed modeled with a process.

`(sos-duration ?sos ?pfrom ?pto)` is the fixed duration of a given SOS from the start port `?pfrom` to the end port `?pto`.

`(sos-transship-cost ?sos ?pfrom ?pto)` is the fixed cost of transshipping containers to the repositioning vessel at the start of an SOS.

A.1.3 Time

We represent the passage of time with a PDDL process. Figure A.1 shows the PDDL for the process, which uses the dummy precondition `can-start-time` which is true in the initial state. This ensures that the process is executing from time 0, and does not stop until the end of the plan. The process updates `time-elapsed` at each unit of time by a single unit.

A.1.4 Initial and Goal States

As discussed in Section 5.2.1, our model uses TILs to encode the times when activities are allowed to occur, such as port calls or SOS opportunities. Each PDDL time unit represents an hour of repositioning time.

The phase-out service port calls of a vessel, i.e., the times at which it could leave that service and be repositioned, are encoded using the predicate `(vessel-may-phase-out ?vessel ?port)`. If a vessel v can phase out at at time t from port p, the relevant fact is added as a TIL at time t, and subsequently deleted as a TIL shortly afterwards, to reflect the fact that if the opportunity is not taken then, the vessel will continue on its current service.

Similar to our use of TILs for the phase-out, we indicate what time slots are available at what ports using a TIL combined with the predicate `(vessel-may-phase-in ?portpi)`. For instance, if one option is to start a service that calls at some port, once a week, at mid-day on a Tuesday, then a TIL for this port will be added once per week at time points corresponding to this. The TIL is deleted shortly afterwards.

In order to ensure that vessels phase-in one after the other with a weekly temporal spacing we use the TIL `(phasein-week-open)`, that triggers at the start of each week, and any vessel that phases-in in that particular week, uses this fact in order to do so. Thus, only one vessel can phase-in each week.

To encode SOS opportunities, we use the predicate `(sos-open ?sos ?pfrom ?pto)`. The fact denoting that an SOS opportunity is available is added by a TIL at the appropriate time, and deleted by another shortly afterwards. Thus, if a vessel is to sail

```
1 (:action phase-out
2   :parameters (?vessel - vessel ?port - port)
3   :precondition (and
4     (on-init-service ?vessel)
5     (vessel-may-phase-out ?vessel ?port)
6     (allowed-to-phase-out ?vessel) )
7   :effect (and
8     (not (on-init-service ?vessel))
9     (not (allowed-to-phase-out ?vessel))
10    (can-sail ?vessel)
11    (in-transit ?vessel)
12    (vessel-at ?vessel ?port) ) )
```

Figure A.2: The `phase-out` action.

on a service, it must do so then, and only then.

A.1.5 Actions

We now describe all of the ungrounded actions available to the PDDL model.

Phase-out

The phase out action, shown in Figure A.2, is parameterised by a vessel `?vessel` and a port `?port`. The preconditions (lines 3 – 6) allow the action to be applied if *i*) the vessel is on its initial service, *ii*) the vessel is at `?port` at the time the phase-out is supposed to happen, *iii*) and the hotel cost action has been started. The action immediately deletes the first two facts from the plan (lines 8 and 9) to indicate that the vessel is now performing its repositioning. We then add facts to *i*) indicate that the vessel is allowed to undertake sailing actions, *ii*) set the state of the vessel to "in-transit", meaning its repositioning has started, *iii*) and that the vessel's current position is `?port`.

Phase-in

In order to ensure that vessels phase in to the goal service in subsequent weeks, we split the `phase-in` action in to two actions: `phase-in-1st` and `phase-in-block`. In this modeling construction, a vessel first uses `phase-in-1st`, which sets the port and week of the first phase in. Once this action has been used, other vessels may phase in using `phase-in-block` in a block of weeks subsequent to the week set in `phase-in-1st`, where the size of the block of weeks is equal to the number of vessels.

There are multiple possibilities for modeling the phase-in. Another way would be to pass the phase-in week into the model as a parameter, and put bounds on the earliest and latest phase-in week, or require that the week is subsequent to the week of the last phased-in vessel. Preliminary tests with such models resulted in extremely poor performance, due to the number of extra actions introduced through grounding. Alternatively, one could model phase-ins using a single action with conditional effects

```
1 (:action phase-in-1st
2   :parameters (?portpi - port ?vessel - vessel)
3   :precondition (and
4     (block-phase-in-start)
5     (in-transit ?vessel)
6     (vessel-at ?vessel ?portpi)
7     (vessel-may-phase-in ?portpi)
8     (phasein-week-open))
9   :effect (and
10    (not (phasein-week-open))
11    (not (block-phase-in-start))
12    (not (in-transit ?vessel))
13    (phased-in ?vessel)
14    (allowed-to-end-cost-calc ?vessel)
15    (first-phasein-port ?portpi)
16    (first-phasein-week-defined)
17    (when (and (>= (+ (time-elapsed) (fake-duration)) 0)
18      (< (+ (time-elapsed) (fake-duration)) 168) )
19      (assign (first-phasein-week) 0) )
20    (when (and (>= (+ (time-elapsed) (fake-duration)) 168)
21      (< (+ (time-elapsed) (fake-duration)) 336) )
22      (assign (first-phasein-week) 1) )
23    ...
24    (when (and (>= (+ (time-elapsed) (fake-duration)) 1680)
25       (< (+ (time-elapsed) (fake-duration)) 1848) )
26       (assign (first-phasein-week) 10) ) ) )
```

Figure A.3: The `phase-in-1st` action.

and disjunctive preconditions, which are part of the PDDL 2.2 standard. However, POPF's support for conditional effects is limited, and not sufficient for a single phase-in action.

The `phase-in-1st` action is shown in Figure A.3. We discuss the preconditions on lines 4 – 8 in order. In order to be applied, the action requires that the block-phase in has not yet started, the vessel is in transit, the vessel has sailed to the phase-in port `?portpi`, the vessel is allowed to phase-in (from a temporal stand point), and the week the vessel is to phase-in must be open. The effects of the action (lines 9 – 26) are as follows. First, the action deletes three facts to ensure that the week being used is not open for a phase-in from another vessel, to indicate that the block phase-in has been used, and to set the state of the vessel to be no longer in transit. The vessel state is then set to be phased-in, indicating the vessel is done repositioning. The `(allowed-to-end-cost-calc ?vessel)` predicate indicates that the hotel period for the vessel is over. The first vessel that arrives at the goal service determines which port is used for the entire phase-in, which is set in the `(first-phasein-port ?portpi)` effect, and the first phase-in week is also defined. Finally, for each week we have a conditional effect that assigns the week of the phase-in based on the current time (`time-elapsed`). We repeat this for 10 weeks, which limits the model to a 10 week period. This is sufficient for the purposes of this model.

The `phase-in-block` action in Figure A.4 is similar to the `phase-in-1st` action in many ways, with the exception that it requires that a vessel has used the `phase-in-1st`

```
1 (:action phase-in-block
2   :parameters ( ?vessel - vessel ?portpi - port )
3   :precondition (and
4     (in-transit ?vessel)
5     (vessel-at ?vessel ?portpi)
6     (vessel-may-phase-in ?portpi)
7     (phasein-week-open)
8     (>= (time-elapsed) (* (first-phasein-week) 168))
9     (< (time-elapsed)
10      (+ (* (+ (weeks-within) (first-phasein-week)) 168) 168))
11    (first-phasein-week-defined)
12    (first-phasein-port ?portpi) )
13  :effect (and
14    (not (phasein-week-open))
15    (not (in-transit ?vessel))
16    (phased-in ?vessel)
17    (allowed-to-end-cost-calc ?vessel) ) )
```

Figure A.4: The `phase-in-block` action.

to begin the repositioning block. The first three preconditions on lines 3 – 6 are the same as in the `phase-in-1st` action. The action requires `(phasein-week-open)` to be true, which is added by `phase-in-1st`. On lines 8 – 10, the current time is checked to make sure it is larger than the first phase-in week, but smaller than the first phase-in week plus the total number of weeks required for the phase-in. In this way, the block phase-in structure is enforced. Lines 11 and 12 require that the first phase-in week was defined by the `phase-in-1st` action, and that the first phase-in port matches the port this action is attempting to phase-in at, respectively. The effects of the `phase-in-block` action are the same as the `phase-in-1st` action, with the exception that the first phase-in port and week are not assigned, as they were already provided values.

Sail

After the phase-out and before the phase-in, vessels must sail between ports in order to complete the repositioning and undertake cost-saving activities. The `sail` action allows vessels to transit from one port to another. The action is durative with a duration constrained to be between the minimum and maximum sailing time between `?pfrom` and `?pto` (lines 4 – 5). The preconditions (lines 6 – 10) check, in order, whether the vessel is allowed to sail (i.e., it has not just completed a sailing), whether sailing between the two ports is allowed, whether the vessel's state is in transit (i.e., it has phased out), and whether the vessel's current position is at port `?pfrom`. The effects of the action (lines 11 – 17) are to move the vessel from `?pfrom` to `?pto`, prevent the vessel from performing an immediately subsequent sailing, and to increase the cost based on the linear bunker consumption function. Note that `?duration` is a variable provided by the planner indicating the total duration of the action. Preconditions (effects) marked "at start" are checked (performed) at the time point at which the action begins, and preconditions (effects) marked as "at end" are checked (performed) only at the time point at which

```
(:durative-action sail
  :parameters (?vessel - vessel ?pfrom - port ?pto - port)
  :duration (and
    (>= ?duration (min-time-to-sail ?vessel ?pfrom ?pto))
    (<= ?duration (max-time-to-sail ?vessel ?pfrom ?pto)))
  :condition (and
    (at start (can-sail ?vessel))
    (at start (sailing-allowed ?pfrom ?pto))
    (at start (in-transit ?vessel))
    (at start (vessel-at ?vessel ?pfrom)))
  :effect (and
    (at start (not (vessel-at ?vessel ?pfrom)))
    (at end (vessel-at ?vessel ?pto))
    (at start (not (can-sail ?vessel)))
    (at start (increase (total-cost)
      (+ (fixed-sail-cost ?vessel ?pfrom ?pto)
      (* (variable-sail-cost ?vessel ?pfrom ?pto) ?duration))))))
```

Figure A.5: The `sail` action.

```
(:durative-action sail-equipment
  :parameters (?vessel - vessel ?pfrom - port ?pto - port)
  :duration (and
    (>= ?duration (min-time-to-sail ?vessel ?pfrom ?pto))
    (<= ?duration (max-time-to-sail ?vessel ?pfrom ?pto)))
  :condition (and
    (at start (vessel-at ?vessel ?pfrom))
    (at start (equipment-sailing ?pfrom ?pto))
    (at start (sos-or-equipment-allowed ?vessel)) )
  :effect (and
    (at start (not (vessel-at ?vessel ?pto)))
    (at end (vessel-at ?vessel ?pto))
    (at start (not (sos-or-equipment-allowed ?vessel)))
    (at end (can-sail ?vessel))
    (at start (increase (total-cost)
      (+ (fixed-eqp-sail-cost ?vessel ?pfrom ?pto)
      (* (variable-eqp-sail-cost ?vessel ?pfrom ?pto) ?duration))))))
```

Figure A.6: The `sail-equipment` action.

the action completes. This action shows how PDDL handles time-dependent task costs, in which an action effect is used to increase the objective function `total-cost`. We update the objective function at the start of the action due to the way POPF handles such task costs. From a PDDL standpoint, this update could also be at the end of the action.

Sail Equipment

Another activity that vessels may perform while repositioning is to carry equipment from one port to another. Since the NCLSFRP does not model cargo or equipment flows, we model equipment sailing as a cost reduction in sailing. The `sail-equipment` action allows vessels to carry equipment at a reduced bunker consumption cost between ports. Figure A.6 shows the sail equipment action. The duration constraints are the

```
1 (:durative-action sail-on-service
2   :parameters (?vessel - vessel ?pfrom - port ?pto - port ?sos - sos)
3   :duration (= ?duration (sos-duration ?sos ?pfrom ?pto))
4   :condition (and
5     (at start (vessel-at ?vessel ?pfrom))
6     (at start (sos-or-equipment-allowed ?vessel))
7     (at start (sos-open ?sos ?pfrom ?pto))
8     (at start (unused ?sos)) )
9     (at start (in-transit ?vessel))
10    (at end (allowed-to-end-sos ?vessel))
11  :effect (and
12    (at start (not (sos-open ?sos ?pfrom ?pto)))
13    (at start (not (unused ?sos)))
14    (at start (not (sos-or-equipment-allowed ?vessel)))
15    (at start (not (vessel-at ?vessel ?pfrom)))
16    (at start (sailing-on-service ?vessel ?sos))
17    (at start (allowed-to-end-cost-calc ?vessel))
18    (at start (allowed-to-start-cost-calc ?vessel))
19    (at end (not (sailing-on-service ?vessel ?sos)))
20    (at end (vessel-at ?vessel ?pto))
21    (at end (can-sail ?vessel))
22    (at start (increase (total-cost) (sos-hotel-cost ?sos ?pfrom ?pto)))))
```

Figure A.7: The `sail-on-service` action.

same as for the sail action, however the preconditions on lines 6 – 9 do not require that the `can-sail` predicate be available, as sailing and then sailing with equipment is not considered a sequential sailing. The preconditions require that a sail equipment action is allowed between `?pfrom` and `?pto`, i.e. `?pfrom` has an equipment surplus and `?pto` a demand for equipment. The vessel is prevented from chaining SOS and sail equipment opportunities in the final precondition. The effects first move the vessel from `?pfrom` to `?pto`, followed by preventing SOS and sail equipment action chaining. The vessel is allowed to perform a sail action after the sail equipment activity on line 14, and finally the cost of the sailing is updated with sail equipment specific bunker consumption coefficients.

Sail on Service

The `sail-on-service` action shown in Figure A.7 allows vessels to move between ports for little to no cost. As in the case of the `sail-equipment` action, the vessel must be at the from port and not have performed a different SOS or sail equipment previously. On lines 4 – 9, it is ensured that there is actually an SOS from `?pfrom` to `?pto`, that the SOS is not being used by any other vessel, and that the current vessel has phased-out. The last precondition on line 10 is used to help calculate the hotel cost, since over the period of the SOS no hotel cost is incurred by the vessel using the SOS. The effects at the start of the action, lines 11 – 18, close the SOS to other vessels, mark the SOS as being used, prevent SOS and sail equipment action chaining, mark the vessel as no longer being at `?pfrom`, indicate that the vessel is on the SOS, and regulate the end of

```
1 (:durative-action hotel-cost-calc-phase-out
2   :parameters (?vessel - vessel)
3   :duration (and (>= ?duration 0.01) (<= ?duration 10000))
4   :condition (and
5     (at start (on-init-service ?vessel))
6     (at start (cost-calc-mutex ?vessel))
7     (at start (allowed-to-start-cost-calc ?vessel))
8     (at end (allowed-to-end-cost-calc ?vessel)) )
9   :effect (and
10    (at start (not (allowed-to-start-cost-calc ?vessel)))
11    (at start (not (cost-calc-mutex ?vessel)))
12    (at start (allowed-to-phase-out ?vessel))
13    (at end (not (allowed-to-end-cost-calc ?vessel)))
14    (at end (cost-calc-mutex ?vessel))
15    (at end (hotel-cost-calculated ?vessel))
16    (at start (increase (total-cost)(* (hotel-cost ?vessel) ?duration)))))
```

Figure A.8: The `hotel-cost-calc-phase-out` action.

the previous hotel period and the start of the next hotel period, respectively. At the end of the action, lines 19 – 21, the vessel is marked as having left the SOS, assigned the location `?pto`, and the vessel is allowed to perform a `sail` action. Finally, the cost of using the SOS is increased by a small amount.

Hotel Cost

The hotel cost is computed using an envelope action, which means that the action encloses a number of other actions over its duration. In the case of computing the hotel cost, the envelope action encloses all repositioning activities (except for the SOS) between the phase-out and phase-in.

The hotel cost is computed using two actions, `hotel-cost-calc-phase-out`, which is starts the hotel period at the phase-out and continues until either a phase-in or an SOS, and `hotel-cost-calc-sos`, which restarts the hotel period after an SOS. Figure A.8 shows the `hotel-cost-calc-phase-out` action. The action requires that the vessel is on its initial service, that no other hotel period is currently occurring at the same time as this one for this vessel, and that the hotel period covers the entire applicable time span. The `allowed-to-end-cost-calc` predicate is only added at the end of the hotel period, ensuring the duration of the hotel cost spans the entire relevant part of the repositioning. The effects of the action are to negate the `allowed-to-end-cost-calc` predicate so another period could be used after an SOS if necessary, prevent any other hotel period from happening at the same time as this one, allow the vessel to phase out, and indicate that the hotel cost has been calculated (for the goal state). Finally, the hotel cost is calculated over the duration of the action.

The `hotel-cost-calc-sos` action shares the same preconditions as `hotel-cost-calc-phase-out`, except that the `on-init-service` predicate is replaced with the `sailing-on-service` predicate, as the action only starts after an SOS takes place. The effects of the action are also the same as `hotel-cost-calc-sos`, except that in-

stead of allowing the vessel to phase-out, the vessel is allowed to end its SOS with the `allowed-to-end-sos` predicate.

While future planners may be able to handle a single hotel cost action, current planners require this split of activities in order to have a "hole" in the hotel cost period. For example, PDDL would support the following computation of hotel cost:

```
(increase (total-cost) (* (hotel-cost ?vessel) (- ?duration (sos-duration ?vessel))))
```

With this modification to the total plan cost, the time used for an SOS is subtracted from the duration of the hotel period. This type of modeling is difficult for planners to deal with, thus we simply split the hotel cost period in two when necessary.

A.2 Forward PDDL Domain

```
(define (domain fleetrepos)
  (:requirements :typing :durative-actions :fluents :duration-inequalities :action-costs :timed-initial-literals :conditional-effects :time)
  (:types
      sos - object
      port locatable - object
      vessel obj - locatable)
  (:predicates
    (on-init-service ?v - vessel)
    (in-transit ?v - vessel)
    (phased-in ?v - vessel)
    (vessel-at ?obj - locatable ?loc - port)
    (sailing-allowed ?pfrom ?pto - port)
    (equipment-sailing ?pfrom ?pto - port)
    (sos-open ?sos - sos ?pfrom ?pto - port)
    (sos-or-equipment-allowed ?vessel - vessel)
    (used ?s - sos)
    (unused ?s - sos)
    (sailing-on-service ?v - vessel ?s - sos)
    (vessel-may-phase-out ?v - vessel ?p - port)
    (vessel-may-phase-in ?p - port)
    (hotel-cost-calculated ?v - vessel)
    (cost-calc-mutex ?v - vessel)
    (phasein-week-open)
    (first-phasein-port ?port - port)
    (first-phasein-week-defined)
    (block-phase-in-start)
    (can-start-time)
    (allowed-to-phase-out ?v - vessel)
    (allowed-to-start-cost-calc ?v - vessel)
    (allowed-to-end-cost-calc ?v - vessel)
    (allowed-to-end-sos ?v - vessel)
    (not-moved ?v - vessel)
    (can-sail ?v - vessel) )

(:functions
    (min-time-to-sail ?vessel - vessel ?pfrom ?pto - port)
    (max-time-to-sail ?vessel - vessel ?pfrom ?pto - port)
    (sos-transship-cost ?sos - sos ?pfrom ?pto - port)
    (sos-duration ?sos - sos ?pfrom ?pto - port)
    (total-cost)
    (fixed-sail-cost ?v - vessel ?pfrom ?pto - port)
    (fixed-eqp-sail-cost ?v - vessel ?pfrom ?pto - port)
    (variable-sail-cost ?v - vessel ?pfrom ?pto - port)
    (variable-eqp-sail-cost ?v - vessel ?pfrom ?pto - port)
```

```
45      (hotel-cost ?v - vessel) ; hotel cost per hour for v
46      (time-elapsed)
47      (first-phasein-week)
48      (weeks-within)
49      (fake-duration)
50      (number-sos) )
51
52  (:process time-is-passing
53      :parameters ()
54      :precondition  (can-start-time)
55      :effect (increase (time-elapsed) (* #t 1.0) ) )
56
57  (:durative-action HOTEL-COST-CALC-PHASE-OUT
58      :parameters
59       ( ?vessel - vessel )
60      :duration (and (>= ?duration 0.01) (<= ?duration 10000))
61      :condition
62      (and
63          (at start (on-init-service ?vessel))
64          (at start (cost-calc-mutex ?vessel))
65          (at start (allowed-to-start-cost-calc ?vessel))
66          (at end (allowed-to-end-cost-calc ?vessel)) )
67      :effect
68      (and
69          (at start (increase (total-cost) (* (hotel-cost ?vessel) ?duration)))
70          (at start (allowed-to-phase-out ?vessel))
71          (at end (hotel-cost-calculated ?vessel))
72          (at end (not (allowed-to-end-cost-calc ?vessel)))
73          (at start (not (allowed-to-start-cost-calc ?vessel)))
74          (at start (not (cost-calc-mutex ?vessel)))
75          (at end (cost-calc-mutex ?vessel)) ) )
76
77  (:durative-action HOTEL-COST-CALC-SOS
78      :parameters
79       ( ?vessel - vessel
80         ?sos - sos )
81      :duration (and (>= ?duration 0.01) (<= ?duration 10000))
82      :condition
83      (and
84          (at start (sailing-on-service ?vessel ?sos))
85          (at start (allowed-to-start-cost-calc ?vessel))
86          (at end (allowed-to-end-cost-calc ?vessel))
87          (at start (cost-calc-mutex ?vessel)) )
88      :effect
89      (and
90          (at start (increase (total-cost) (* (hotel-cost ?vessel) ?duration)))
91          (at start (allowed-to-end-sos ?vessel))
92          (at end (not (allowed-to-end-cost-calc ?vessel)))
93          (at start (not (allowed-to-start-cost-calc ?vessel)))
94          (at start (not (cost-calc-mutex ?vessel)))
95          (at end (cost-calc-mutex ?vessel)) ) )
96
97  (:action PHASE-OUT
98      :parameters
99      (   ?vessel - vessel
100         ?port - port )
101     :precondition
102     (and
103         (on-init-service ?vessel)
104         (vessel-may-phase-out ?vessel ?port)
105         (allowed-to-phase-out ?vessel) )
106     :effect
107     (and
108         (can-sail ?vessel)
109         (not (on-init-service ?vessel))
```

```
110         (in-transit ?vessel)
111         (vessel-at ?vessel ?port)
112         (not (allowed-to-phase-out ?vessel)) ) )
113
114 (:durative-action SAIL-ON-SERVICE
115     :parameters
116     (   ?vessel - vessel
117         ?pfrom - port
118         ?pto - port
119         ?sos - sos )
120     :duration (= ?duration (sos-duration ?sos ?pfrom ?pto))
121     :condition
122     (and
123         (at start (sos-open ?sos ?pfrom ?pto))
124         (at start (in-transit ?vessel))
125         (at start (vessel-at ?vessel ?pfrom))
126         (at start (sos-or-equipment-allowed ?vessel))
127         (at end (allowed-to-end-sos ?vessel))
128         (at start (unused ?sos)) )
129     :effect
130     (and
131         (at start (sailing-on-service ?vessel ?sos))
132         (at end (not (sailing-on-service ?vessel ?sos)))
133         (at start (allowed-to-end-cost-calc ?vessel))
134         (at start (allowed-to-start-cost-calc ?vessel))
135         (at start (not (sos-open ?sos ?pfrom ?pto)))
136         (at start (not (vessel-at ?vessel ?pfrom)))
137         (at start (not (sos-or-equipment-allowed ?vessel)))
138         (at start (not (unused ?sos)))
139         (at end (vessel-at ?vessel ?pto))
140         (at start (increase (total-cost) (sos-transship-cost ?sos ?pfrom ?pto)))
141         (at end (used ?sos))
142         (at start (decrease (number-sos) 1))
143         (at start (not (not-moved ?vessel)))
144         (at end (can-sail ?vessel)) ) )
145
146 (:action PHASE-IN-1ST
147  :parameters (?portpi - port ?vessel - vessel)
148  :precondition (and
149     (block-phase-in-start)
150     (in-transit ?vessel)
151     (vessel-at ?vessel ?portpi)
152     (vessel-may-phase-in ?portpi)
153     (phasein-week-open) )
154  :effect (and
155     (not (phasein-week-open))
156     (not (in-transit ?vessel))
157     (phased-in ?vessel)
158     (allowed-to-end-cost-calc ?vessel)
159
160     (first-phasein-port ?portpi)
161     (not (block-phase-in-start))
162     (first-phasein-week-defined)
163     (when (and (>= (+ (time-elapsed) (fake-duration)) 0)
164                (< (+ (time-elapsed) (fake-duration)) 168) )
165           (assign (first-phasein-week) 0) )
166     (when (and (>= (+ (time-elapsed) (fake-duration)) 168)
167                (< (+ (time-elapsed) (fake-duration)) 336) )
168           (assign (first-phasein-week) 1) )
169     (when (and (>= (+ (time-elapsed) (fake-duration)) 336)
170                (< (+ (time-elapsed) (fake-duration)) 504) )
171           (assign (first-phasein-week) 2) )
172     (when (and (>= (+ (time-elapsed) (fake-duration)) 504)
173                (< (+ (time-elapsed) (fake-duration)) 672) )
174           (assign (first-phasein-week) 3) )
```

```
175     (when (and (>= (+ (time-elapsed) (fake-duration)) 672)
176                (< (+ (time-elapsed) (fake-duration)) 840) )
177           (assign (first-phasein-week) 4) )
178     (when (and (>= (+ (time-elapsed) (fake-duration)) 840)
179                (< (+ (time-elapsed) (fake-duration)) 1008) )
180           (assign (first-phasein-week) 5) )
181     (when (and (>= (+ (time-elapsed) (fake-duration)) 1008)
182                (< (+ (time-elapsed) (fake-duration)) 1176) )
183           (assign (first-phasein-week) 6) )
184     (when (and (>= (+ (time-elapsed) (fake-duration)) 1176)
185                (< (+ (time-elapsed) (fake-duration)) 1344) )
186           (assign (first-phasein-week) 7) )
187     (when (and (>= (+ (time-elapsed) (fake-duration)) 1344)
188                (< (+ (time-elapsed) (fake-duration)) 1512) )
189           (assign (first-phasein-week) 8) )
190     (when (and (>= (+ (time-elapsed) (fake-duration)) 1512)
191                (< (+ (time-elapsed) (fake-duration)) 1680) )
192           (assign (first-phasein-week) 9) )
193     (when (and (>= (+ (time-elapsed) (fake-duration)) 1680)
194                (< (+ (time-elapsed) (fake-duration)) 1848) )
195           (assign (first-phasein-week) 10) ) ) )
196
197 (:durative-action SAIL-EQUIPMENT
198     :parameters
199     (   ?vessel - vessel
200         ?pfrom - port
201         ?pto - port )
202     :duration (and (>= ?duration (min-time-to-sail ?vessel ?pfrom ?pto))
203                    (>= ?duration 0.01)
204                    (<= ?duration (max-time-to-sail ?vessel ?pfrom ?pto)) )
205     :condition
206     (and
207         (at start (vessel-at ?vessel ?pfrom))
208         (at start (equipment-sailing ?pfrom ?pto))
209         (at start (sos-or-equipment-allowed ?vessel)) )
210     :effect
211     (and
212         (at end (can-sail ?vessel))
213         (at start (not (vessel-at ?vessel ?pto)))
214         (at start (not (sos-or-equipment-allowed ?vessel)))
215         (at end (vessel-at ?vessel ?pto))
216         (at start (increase (total-cost) (+ (fixed-eqp-sail-cost ?vessel ?pfrom ?pto)
                (* (variable-eqp-sail-cost ?vessel ?pfrom ?pto) ?duration)))) ) )
217
218 (:action PHASE-IN-BLOCK
219     :parameters
220     (   ?vessel - vessel
221         ?portpi - port )
222     :precondition
223     (and
224         (phasein-week-open)
225         (in-transit ?vessel)
226         (vessel-at ?vessel ?portpi)
227         (vessel-may-phase-in ?portpi)
228         (>= (time-elapsed) (* (first-phasein-week) 168))
229         (< (time-elapsed)  (+ (* (+ (weeks-within) (first-phasein-week)) 168) 168))
230         (first-phasein-week-defined)
231         (first-phasein-port ?portpi) )
232     :effect
233     (and
234         (not (vessel-at ?vessel ?portpi))
235         (not (phasein-week-open))
236         (not (in-transit ?vessel))
237         (phased-in ?vessel)
238         (allowed-to-end-cost-calc ?vessel) ) )
```

```
239
240 (:durative-action SAIL
241     :parameters
242     (   ?vessel - vessel
243         ?pfrom - port
244         ?pto - port )
245     :duration (and (>= ?duration (min-time-to-sail ?vessel ?pfrom ?pto))
246                    (<= ?duration (max-time-to-sail ?vessel ?pfrom ?pto)) )
247     :condition
248     (and
249         (at start (can-sail ?vessel))
250         (at start (sailing-allowed ?pfrom ?pto))
251         (at start (in-transit ?vessel))
252         (at start (vessel-at ?vessel ?pfrom)) )
253     :effect
254     (and
255         (at start (not (vessel-at ?vessel ?pfrom)))
256         (at start (not (can-sail ?vessel)))
257         (at end (vessel-at ?vessel ?pto))
258         (at start (increase (total-cost) (+ (fixed-sail-cost ?vessel ?pfrom ?pto) (* (
            variable-sail-cost ?vessel ?pfrom ?pto) ?duration)))) ) ) )
```

A.3 Reversed PDDL Domain

```
1  (define (domain fleetrepos)
2      (:requirements :typing :durative-actions :fluents :duration-inequalities :action-
           costs :timed-initial-literals :conditional-effects :time)
3      (:types
4        sos - object
5        port locatable - object
6        vessel obj - locatable)
7    (:predicates
8      (on-init-service ?v - vessel)
9      (in-transit ?v - vessel)
10     (phased-in ?v - vessel)
11     (vessel-at ?obj - locatable ?loc - port)
12     (sailing-allowed ?pfrom ?pto - port)
13     (vessel-may-sail ?vessel - vessel)
14     (equipment-sailing ?pfrom ?pto - port)
15     (sos-open ?sos - sos ?pfrom ?pto - port)
16     (sos-or-equipment-allowed ?vessel - vessel)
17     (used ?s - sos)
18     (unused ?s - sos)
19     (sailing-on-service ?v - vessel ?s - sos)
20     (sos-closes ?s - sos ?pfrom - port ?pto - port)
21     (vessel-may-phase-out ?v - vessel ?p - port)
22     (vessel-may-phase-in ?p - port)
23     (hotel-cost-calculated ?v - vessel)
24     (cost-calc-mutex ?v - vessel)
25     (phasein-week-open)
26     (first-phasein-port ?port - port)
27     (first-phasein-week-defined)
28     (block-phase-in-start)
29     (can-start-time)
30     (allowed-to-phase-in ?v - vessel)
31     (allowed-to-phase-out ?v - vessel)
32     (allowed-to-start-cost-calc ?v - vessel)
33     (allowed-to-end-cost-calc ?v - vessel)
34     (allowed-to-end-sos ?v - vessel)
35     (not-moved ?v - vessel)
36     (can-sail ?v - vessel) )
37
38 (:functions
39     (min-time-to-sail ?vessel - vessel ?pfrom ?pto - port)
```

```
    (max-time-to-sail ?vessel - vessel ?pfrom ?pto - port)
    (sos-transship-cost ?sos - sos ?pfrom ?pto - port)
    (sos-duration ?sos - sos ?pfrom ?pto - port)
    (total-cost)
    (fixed-sail-cost ?v - vessel ?pfrom ?pto - port)
    (fixed-eqp-sail-cost ?v - vessel ?pfrom ?pto - port)
    (variable-sail-cost ?v - vessel ?pfrom ?pto - port)
    (variable-eqp-sail-cost ?v - vessel ?pfrom ?pto - port)
    (hotel-cost ?v - vessel)
    (time-elapsed)
    (first-phasein-week)
    (weeks-within)
    (fake-duration)
    (number-vessels)
    (number-sos) )

(:process time-is-passing
    :parameters ()
    :precondition  (can-start-time)
    :effect (increase (time-elapsed) (* #t 1.0) ) )

(:durative-action HOTEL-COST-CALC-PHASE-IN-REVERSE
    :parameters
     ( ?vessel - vessel )
    :duration (and (>= ?duration 0.01) (<= ?duration 10000))
    :condition
    (and
        (at start (phased-in ?vessel))
        (at start (cost-calc-mutex ?vessel))
        (at start (allowed-to-start-cost-calc ?vessel))
        (at end (allowed-to-end-cost-calc ?vessel)) )
    :effect
    (and
        (at start (increase (total-cost) (* (hotel-cost ?vessel) ?duration)))
        (at start (allowed-to-phase-in ?vessel))
        (at end (hotel-cost-calculated ?vessel))
        (at end (not (allowed-to-end-cost-calc ?vessel)))
        (at start (not (allowed-to-start-cost-calc ?vessel)))
        (at start (not (cost-calc-mutex ?vessel)))
        (at end (cost-calc-mutex ?vessel)) ) )

(:durative-action HOTEL-COST-CALC-SOS-REVERSE
    :parameters
     ( ?vessel - vessel
       ?sos - sos )
    :duration (and (>= ?duration 0.01) (<= ?duration 10000))
    :condition
    (and
        (at start (sailing-on-service ?vessel ?sos))
        (at start (allowed-to-start-cost-calc ?vessel))
        (at end (allowed-to-end-cost-calc ?vessel))
        (at start (cost-calc-mutex ?vessel)) )
    :effect
    (and
        (at start (increase (total-cost) (* (hotel-cost ?vessel) ?duration)))
        (at start (allowed-to-end-sos ?vessel))
        (at end (not (allowed-to-end-cost-calc ?vessel)))
        (at start (not (allowed-to-start-cost-calc ?vessel)))
        (at start (not (cost-calc-mutex ?vessel)))
        (at end (cost-calc-mutex ?vessel)) ) )

(:action PHASE-OUT-REVERSE
    :parameters
    (   ?vessel - vessel
        ?port - port )
```

```
105     :precondition
106     (and
107         (vessel-may-phase-out ?vessel ?port)
108         (vessel-at ?vessel ?port)
109         (in-transit ?vessel) )
110     :effect
111     (and
112         (not (can-sail ?vessel))
113         (on-init-service ?vessel)
114         (not (in-transit ?vessel))
115         (not (vessel-at ?vessel ?port))
116         (allowed-to-phase-out ?vessel)
117         (allowed-to-end-cost-calc ?vessel) ) )
118
119 (:durative-action SAIL-ON-SERVICE-REVERSE
120     :parameters
121     (   ?vessel - vessel
122         ?pfrom - port
123         ?pto - port
124         ?sos - sos )
125     :duration (= ?duration (sos-duration ?sos ?pfrom ?pto))
126     :condition
127     (and
128         (at start (sos-closes ?sos ?pfrom ?pto))
129         (at start (in-transit ?vessel))
130         (at start (vessel-at ?vessel ?pto))
131         (at start (sos-or-equipment-allowed ?vessel))
132         (at end (allowed-to-end-sos ?vessel))
133         (at start (unused ?sos)) )
134     :effect
135     (and
136         ;(at end (vessel-may-sail ?vessel))
137         (at start (sailing-on-service ?vessel ?sos))
138         (at end (not (sailing-on-service ?vessel ?sos)))
139         (at start (allowed-to-end-cost-calc ?vessel))
140         (at start (allowed-to-start-cost-calc ?vessel))
141         (at start (not (sos-open ?sos ?pfrom ?pto)))
142         (at start (not (vessel-at ?vessel ?pto)))
143         (at start (not (sos-or-equipment-allowed ?vessel)))
144         (at start (not (unused ?sos)))
145         (at end (vessel-at ?vessel ?pfrom))
146         (at start (increase (total-cost) (sos-transship-cost ?sos ?pfrom ?pto)))
147         (at end (used ?sos))
148         (at start (decrease (number-sos) 1))
149         (at start (not (not-moved ?vessel)))
150         (at end (can-sail ?vessel)) ) )
151
152 (:action PHASE-IN-1ST-REVERSE
153  :parameters (?portpi - port ?vessel - vessel)
154  :precondition (and
155     (block-phase-in-start)
156     (phased-in ?vessel)
157     (vessel-may-phase-in ?portpi)
158     (phasein-week-open)
159     (allowed-to-phase-in ?vessel) )
160  :effect (and
161     (vessel-at ?vessel ?portpi)
162     (in-transit ?vessel)
163     (not (phased-in ?vessel))
164     (not (phasein-week-open))
165     (first-phasein-port ?portpi)
166     (not (block-phase-in-start))
167     (first-phasein-week-defined)
168     (can-sail ?vessel)
169     (when (and (>= (+ (time-elapsed) (fake-duration)) 0)
```

```
                (< (+ (time-elapsed) (fake-duration)) 168))
            (assign (first-phasein-week) 0))
    (when (and (>= (+ (time-elapsed) (fake-duration)) 168)
                (< (+ (time-elapsed) (fake-duration)) 336))
            (assign (first-phasein-week) 1))
    (when (and (>= (+ (time-elapsed) (fake-duration)) 336)
                (< (+ (time-elapsed) (fake-duration)) 504))
            (assign (first-phasein-week) 2))
    (when (and (>= (+ (time-elapsed) (fake-duration)) 504)
                (< (+ (time-elapsed) (fake-duration)) 672))
            (assign (first-phasein-week) 3))
    (when (and (>= (+ (time-elapsed) (fake-duration)) 672)
                (< (+ (time-elapsed) (fake-duration)) 840))
            (assign (first-phasein-week) 4))
    (when (and (>= (+ (time-elapsed) (fake-duration)) 840)
                (< (+ (time-elapsed) (fake-duration)) 1008))
            (assign (first-phasein-week) 5))
    (when (and (>= (+ (time-elapsed) (fake-duration)) 1008)
                (< (+ (time-elapsed) (fake-duration)) 1176))
            (assign (first-phasein-week) 6))
    (when (and (>= (+ (time-elapsed) (fake-duration)) 1176)
                (< (+ (time-elapsed) (fake-duration)) 1344))
            (assign (first-phasein-week) 7))
    (when (and (>= (+ (time-elapsed) (fake-duration)) 1344)
                (< (+ (time-elapsed) (fake-duration)) 1512))
            (assign (first-phasein-week) 8))
    (when (and (>= (+ (time-elapsed) (fake-duration)) 1512)
                (< (+ (time-elapsed) (fake-duration)) 1680))
            (assign (first-phasein-week) 9))
    (when (and (>= (+ (time-elapsed) (fake-duration)) 1680)
                (< (+ (time-elapsed) (fake-duration)) 1848))
            (assign (first-phasein-week) 10)) ) )

(:durative-action SAIL-EQUIPMENT
    :parameters
    (   ?vessel - vessel
        ?pfrom - port
        ?pto - port )
    :duration (and (>= ?duration (min-time-to-sail ?vessel ?pfrom ?pto))
                   (>= ?duration 0.01)
                   (<= ?duration (max-time-to-sail ?vessel ?pfrom ?pto)) )
    :condition
    (and
        (at start (vessel-at ?vessel ?pto))
        (at start (equipment-sailing ?pfrom ?pto))
        (at start (sos-or-equipment-allowed ?vessel)) )
    :effect
    (and
        (at end (can-sail ?vessel))
        (at start (not (vessel-at ?vessel ?pto)))
        (at start (not (sos-or-equipment-allowed ?vessel)))
        (at end (vessel-at ?vessel ?pfrom))
        (at start (increase (total-cost) (+ (fixed-eqp-sail-cost ?vessel ?pfrom ?pto)
            (* (variable-eqp-sail-cost ?vessel ?pfrom ?pto) ?duration)))) ) )

(:action PHASE-IN-BLOCK-REVERSE
    :parameters
    (   ?vessel - vessel
        ?portpi - port )
    :precondition
    (and
        (phasein-week-open)
        (phased-in ?vessel)
        (vessel-may-phase-in ?portpi)
        (>= (time-elapsed) (* (first-phasein-week) 168))
```

```
234         (< (time-elapsed)  (+ (* (+ (weeks-within) (first-phasein-week)) 168) 168))
235         (first-phasein-week-defined)
236         (first-phasein-port ?portpi)
237         (allowed-to-phase-in ?vessel) )
238     :effect
239     (and
240         (vessel-at ?vessel ?portpi)
241         (in-transit ?vessel)
242         (not (phasein-week-open))
243         (not (phased-in ?vessel))
244         (can-sail ?vessel) ) )
245
246 (:durative-action SAIL-REVERSE
247     :parameters
248     (   ?vessel - vessel
249         ?pfrom - port
250         ?pto - port )
251     :duration (and (>= ?duration (min-time-to-sail ?vessel ?pfrom ?pto))
252                    (<= ?duration (max-time-to-sail ?vessel ?pfrom ?pto)) )
253     :condition
254     (and
255         ;(at start (vessel-may-sail ?vessel))
256         (at start (can-sail ?vessel))
257         (at start (sailing-allowed ?pfrom ?pto))
258         (at start (in-transit ?vessel))
259         (at start (vessel-at ?vessel ?pto)) )
260     :effect
261     (and
262         (at start (not (vessel-at ?vessel ?pto)))
263         ;(at start (not (vessel-may-sail ?vessel)))
264         (at start (not (can-sail ?vessel)))
265         (at end (vessel-at ?vessel ?pfrom))
266         (at start (increase (total-cost) (+ (fixed-sail-cost ?vessel ?pfrom ?pto) (* (
            variable-sail-cost ?vessel ?pfrom ?pto) ?duration)))) ) ) )
```

Appendix B

An LTOP Model of Fleet Repositioning

The LTOP model consists of six actions: `sail`, `sail-equipment`, `sail-on-service`, `phase-out`, `phase-in`, `phase-in-sos`. Unlike in the PDDL model, LTOP does not require any envelope actions to represent the hotel period. It does, however, have to split the phase-in into two different actions in order to correctly calculate the hotel cost when an SOS is present. We now describe each of these actions, along with the optimization variables and state variables LTOP uses.

B.1 Constants

The following constants are used in the model and are set in the initial state. They can be thought of as functions in the PDDL model. We use the following sets to help describe the constants. Let P be the set of all ports, $P^{PI} \subset P$ be the set of ports on the phase-in (i.e., goal) service, V be the set of vessels, S be the set of of SOSs, and T_p be the set of times in which a phase-in is allowed at port $p \in P^{PI}$. Note that each SOS opportunity consists of multiple SOS actions, one for each start port.

`hotel-cost(ves)` provides the hourly fuel consumption cost of vessel `ves` $\in V$.

`var-sail-cost(ves,pfrom,pto)` provides the variable bunker cost coefficient of vessel `ves` $\in V$ between ports `pfrom,pto` $\in P$.

`fixed-sail-cost(ves,pfrom,pto)` specifies the fixed bunker cost coefficient of vessel `ves` $\in V$ between ports `pfrom,pto` $\in P$.

`var-sail-equip-cost(ves,pfrom,pto)` provides the variable bunker cost coefficient for sail equipment actions of vessel `ves` $\in V$ between ports `pfrom,pto` $\in P$.

`fixed-sail-equip-cost(ves,pfrom,pto)` specifies the fixed bunker cost coefficient for sail equipment actions of vessel `ves` $\in V$ between ports `pfrom,pto` $\in P$.

`min-time(ves,pfrom,pto)` is the minimum time required for vessel `ves` $\in V$ to sail from `pfrom` $\in P$ to `pto` $\in P$.

`max-time(ves,pfrom,pto)` is the maximum time required for vessel `ves` $\in V$ to sail from `pfrom` $\in P$ to `pto` $\in P$.

`fixed-sos-cost(port,sos)` is the fixed cost for using `sos` $\in S$ at `port` $\in P$.

`start-time(port, sos)` is the starting time of `sos` $\in S$ at `port` $\in P$.

`end-time(sos)` is the ending time of `sos` $\in S$.

B.2 State Variables

We introduce the following state variables for our model, using the previously defined constants.

`vessel-state(ves)` $\in \{\text{I}, \text{T}, \text{G}\}$ is the state of the vessel `v` $\in V$, where I means the vessel is on its initial service, T means it is in transit (i.e., repositioning), and G means it is on the goal service.

`vessel-at(ves)` $\in P \cup \{\perp\}$ describes the current port of the vessel, or is set to $\perp$ if the vessel has no location, such as when it has not yet phased out.

`sos-open(sos)` $\in \mathbb{B}$ is true when SOS opportunity `sos` $\in S$ is being used by a vessel.

`using-sos-or-se(ves)` $\in \mathbb{B}$ indicates whether vessel `ves` $\in V$ is using any SOS or sail equipment opportunity.

`pi-open(port,time)` $\in \mathbb{B}$ is true when port `port` $\in P$ at `time` $\in \mathbb{Z}$ is open for a phase-in.

B.3 Optimization Variables

Actions in LTOP are automatically associated with a begin and end time optimization variable which can be used in the objective and constraints of the action. The begin and end time of each action is determined by the variables x^B and x^E, respectively, which we create for each action. Since we present an ungrounded representation of actions, we do not specify the individual begin and end time variables for each ungrounded action. Let $h^B_{\texttt{ves}}$ and $h^E_{\texttt{ves}}$ be the beginning and end times of the hotel period for vessel v, including any time in which an SOS occurs. To handle repositionings containing an SOS, let $\Delta^{SOS}_{\texttt{ves}}$ be the duration of the SOS undertaken for vessel v. If a vessel does not use an SOS, then this variable is set to zero.

B.4 Initial and Goal States

In the initial state, all of the vessels are assigned to have an "initial" (meaning non-phased out) status, all phase-in slots are set to be open, the vessel locations are set to $\perp$ (meaning they are still performing regular duties), all SOS-opportunities are opened,

and all vessels are indicated as having not yet used an SOS or sail equipment opportunity. Formally, we represent the initial state with the following mappings of state variables to values:

$$\begin{aligned}
\texttt{vessel-state}(v) &\leftarrow \textsc{i} && \forall v \in V \\
\texttt{vessel-at}(v) &\leftarrow \bot && \forall v \in V \\
\texttt{using-sos-or-se}(v) &\leftarrow \texttt{false} && \forall v \in V \\
\texttt{sos-open}(s) &\leftarrow \texttt{true} && \forall s \in S \\
\texttt{pi-open}(p,t) &\leftarrow \texttt{true} && \forall p \in P^{PI}, t \in T_p
\end{aligned}$$

The goal state of the LTOP model of the NCLSFRP requires that all vessels have reached the goal state, meaning that they have phased in. Thus, the goal state is represented by the conjunction: $\bigwedge_{v \in V} \texttt{vessel-state}(v) = \textsc{g}$.

B.5 Actions

We describe the actions used in the LTOP model using a custom, pseudo-code syntax. LTOP is implemented as a C++ library, and does not accept PDDL. Furthermore, LTOP does not support an ungrounded representation of actions, which we show for succinctness. Creating the grounded representation of the model involves checking the feasibility of certain actions, such as whether or not sailing between certain ports is feasible. We discuss the non-model preconditions that must hold for each action in order to add it to the model. In each of the ungrounded actions, we will do not parameterize these variables. However, these begin and end time variables are present for every ungrounded action in the model. This is similar to the way the `?duration` variable is used in PDDL models.

B.5.1 Phase-out

The ungrounded `phase-out` action is shown in Figure B.1. We create a grounded phase-out action for each phase-out port at each potential phase-out time for each vessel. In other words, consider the first port the vessel could be at at time 0. We create a phase-out action at that port, substituting the phase-out time (a real value) for `time`. We then create a `phase-out` action at the next port on the service, and so on, until the phase-out time exceeds the maximum time of the model. The `phase-out` action can only be applied for a particular vessel if the vessel has not yet phased-out of its initial service. When the action is applied, the vessel is assigned the port of the `phase-out` action and the state of the vessel is set to be in transit (`t`). Unlike in the PDDL model, we do not require TILs to set the time of the phase-out. Rather, each grounded `phase-out` action is assigned a time, and if the action is selected by the LTOP planner the time is enforced in the constraints. Additionally, the phase-out action sets the value of the hotel

phase−out(ves, port, time)
Pre: vessel-state(ves) = I
Eff: vessel-state(ves) = T ∧ vessel-at(ves) ← port
Obj: 0
Con: $x^B = x^E =$ time ∧ $h^B_{\texttt{ves}} \leq$ time

Figure B.1: The LTOP phase-out action for vessel ves $\in V$ at port $\in P$ at time $\in \mathbb{Z}$.

phase−in(ves, port, time)
Pre: vessel-state(ves) = T ∧ vessel-at(ves) = port
∧ ¬using-sos-or-se(ves) ∧ pi-open(port, time)
Eff: vessel-state(ves) = G ∧ pi-open(port, time) ← false
∧ (pi-open(p, t) ← false, $\forall p \in P^{PI} \setminus \{p\}, t \in T_p$)
∧ (pi-open(port, t) ← false, $\forall t \in \{t' \in T_{\texttt{port}} \mid |t' - t| \geq |V|\}$)
Obj: $(h^E_{\texttt{ves}} - h^B_{\texttt{ves}})\cdot$hotel-cost(ves)
Con: $x^B = x^E =$ time ∧ $h^E_{\texttt{ves}} \geq$ time ∧ $h^E_{\texttt{ves}} \geq h^B_{\texttt{ves}}$ ∧ $h^E_{\texttt{ves}}, h^B_{\texttt{ves}} \geq 0$

Figure B.2: The LTOP phase-in action for vessel ves $\in V$ at port $\in P$ at time $\in \mathbb{Z}$.

B.5.2 Phase-in

Figures B.2 and B.3 show the ungrounded phase-in and phase-in-sos actions, respectively. We perform a manual grounding process for phase-in actions as in the case of phase-out actions, with one key difference. We create a phase-in action for each vessel, port and time that the port could be visited by any vessel. That is, starting from time 0, in each week there is a potential phase-in for each vessel at each port on the phase-in service. Both phase-in and phase-in-sos may be applied when a vessel has sailed to a phase-in port, is in transit, and the phase-in at that port at the particular time chosen is open. The effects of the actions are the same, with both actions setting the vessel to be in the "goal" state, and then the particular phase-in used by the vessel is closed. Next, all phase-ins at other ports are disabled, and finally all phase-ins at the same port with a weekly temporal difference to the current phase-in greater than the number of vessels are disabled. This ensures a block (i.e., weekly) phase-in.

phase−in−sos(ves, port, time)
Pre: vessel-state(ves) = T ∧ vessel-at(ves) = port
∧ using-sos-or-se(ves) ∧ pi-open(port, time)
Eff: vessel-state(ves) = G ∧ pi-open(port, time) ← false
∧ (pi-open(p, t) ← false, $\forall p \in P^{PI} \setminus \{p\}, t \in T_p$)
∧ (pi-open(port, t) ← false, $\forall t \in \{t' \in T_{\texttt{port}} \mid |t' - t| \geq |V|\}$)
Obj: $(h^E_{\texttt{ves}} - h^B_{\texttt{ves}} - \Delta^{SOS}_{\texttt{ves}})\cdot$hotel-cost(ves)
Con: $x^B = x^E =$ time ∧ $h^E_{\texttt{ves}} \geq$ time ∧ $h^E_{\texttt{ves}} \geq h^B_{\texttt{ves}} + \Delta^{SOS}_{\texttt{ves}}$
∧ $h^E_{\texttt{ves}}, h^B_{\texttt{ves}}, \Delta^{SOS}_{\texttt{ves}} \geq 0$

Figure B.3: The LTOP phase-in-sos action for vessel ves $\in V$ at port $\in P$ at time $\in \mathbb{Z}$.

The two actions differ in that `phase-in` may only be applied for a particular vessel if that vessel has not used an SOS. If the vessel has undertaken an SOS, `phase-in-sos` must be used so that the hotel period is correctly calculated. We compute the hotel cost in the objective of the phase-in since in a reverse planner like LTOP, the phase-in actions are the first actions added to a plan, and thus the hotel cost is represented in the cost of a partial plan throughout optimization. In the `phase-in` action, we multiply the total hotel duration by the `hotel-cost` constant for the vessel `ves`. In the `phase-in-sos` action, we subtract the time spent on the SOS ($\Delta^{SOS}_{\texttt{ves}}$) before multiplying by the hotel cost. The constraints of both actions ensure that the phase-in is set to the correct time, that the hotel cost is at least as large as the phase-in time of the vessel, and that the end time of the hotel period is at least as large as the beginning. In the case of `phase-in-sos`, we ensure that the hotel period is at least as large as the beginning time of the period plus the duration of the SOS. Note that with our calculation of the hotel cost in the phase-in actions, along with the updating of the bounds of the hotel period in the phase-out, no envelope action is necessary for modeling the hotel period.

We require two actions for the phase-in in order to subtract the SOS duration from the hotel cost. Consider if we were to model only a single phase-in action with the same objective as the `phase-in-sos` action. When an SOS is included in the model, its sets $\Delta^{SOS}_{\texttt{ves}}$ to the duration of the SOS and the hotel cost is computed correctly. However, when there is no SOS action in the model to set $\Delta^{SOS}_{\texttt{ves}}$, the optimization variable is free to take any value above 0. The cost optimal value is clearly $\Delta^{SOS}_{\texttt{ves}} = h^E_{\texttt{ves}} - h^B_{\texttt{ves}}$, as this (erroneously) eliminates the hotel cost.

The only way to prevent such a situation from occurring is for some action in the plan to constrain $\Delta^{SOS}_{\texttt{ves}}$ to take the value 0 when no SOS is used. If we were to do this in, say, the phase-out or phase-in action, this would cause an infeasibility when an SOS opportunity is utilized, as two different actions would be attempting to set the value of $\Delta^{SOS}_{\texttt{ves}}$. Thus, the only option for correctly computing the hotel cost minus SOS costs is to have another action. One option could be a "no-SOS-present" action that could set a fact required by the goal state that is also set in the SOS action. That is, either an SOS or "no-SOS-present" action would be required in every plan.

Our solution stands in contrast to this approach, in that we avoid unnecessarily increasing the plan length through extra actions. A phase-in action is always required, so our actions allow the planner to make an early commitment to either having an SOS or not having an SOS, and the hotel period can be correctly calculated throughout the plan in both situations.

B.5.3 Sailing

The ungrounded `sail` action is shown in Figure B.4. The action can be applied when the vessel is in transit (that is, it has phased out) and it is at a port `pf`. The action moves the vessel to port `pt`. The vessel incurs a cost computed from the duration of the sailing action using a fixed bunker consumption cost, which is the cost of sailing at maximum speed, and a variable cost based on the duration. The longer the duration,

```
sail(ves, pf, pt)
  Pre: vessel-at(ves) = pf ∧ vessel-state(ves) = T
  Eff: vessel-at(ves) ← pt
  Obj: fixed-sail-cost(ves,pf,pt)
         − (x^E − x^B)·var-sail-cost(ves,pf,pt)
  Con: min-time(ves,pf,pt) ≤ x^E − x^B ≤ max-time(ves,pf,pt)
         ∧ h^B_ves ≤ x^B
```

Figure B.4: The LTOP `sail` action for vessel `ves` $\in V$ from port `pf` $\in P$ to port `pt` $\in P$.

the lower the cost. The duration is constrained to be within the minimum and maximum time for sailing between the two ports. We create a grounded sailing action between all phase-out ports and all SOS start ports, ports with equipment, and phase-in ports. We also create actions between ports with equipment and SOS end ports and the phase-in ports. We update the beginning of the hotel period to be at the start of the sailing action to improve LTOP's lower bound calculation.

Note that in the PDDL model we included facts to avoid the chaining of sailing actions. In LTOP, we do this through a domain-specific heuristic described in Section 5.3.4. We do this because LTOP must search through many infeasible partial plans with such a domain-independent modeling due to its weaker planning heuristics in comparison to POPF.

B.5.4 Sailing with equipment

The ungrounded `sail-equipment` action in Figure B.5 resembles the `sail` action except for two key differences. The first is that `sail-equipment` cannot be chained with other sail-equipment actions or with SOS actions, and thus the `using-sos-or-se` state variable acts a mutex that prevents this. Second, the costs of the `sail-equipment` action are different than the `sail` action, as sailing with equipment is nearly free when the vessel uses its minimum speed. Note that since we make use of the `using-sos-or-se` state variable, which also determines the type of phase-in we can use, we have to set the SOS duration to 0 (i.e., $\Delta^{SOS}_{\texttt{ves}} = 0$).

We create grounded `sail-equipment` actions between all ports with equipment deficits and ports with equipment surpluses, thus, no state variables or constants are needed to determine whether a particular port has equipment; the action will only exist in the grounded state if the equipment is present at `pfrom` and demanded at `pto`.

B.5.5 Sail-on-service

Finally, SOS opportunities are modeled with the `sail-on-service` action shown in Figure B.6 (shown in its ungrounded form). The preconditions and effects are similar to a sailing action, except that, like `sail-equipment`, the `using-sos- or-se` state variable must be false in order to apply the action. We set `using-sos-or-se` to true after the `sail-on-service` action has been applied to prevent the vessel from using a different SOS opportunity, or from carrying equipment.

sail–equipment(ves, pf, pt)
Pre: vessel-at(ves) = pf $\wedge$ vessel-state(ves) = T
$\wedge$ $\neg$using-sos-or-se(ves)
Eff: vessel-at(ves) $\leftarrow$ pt $\wedge$ using-sos-or-se(ves) $\leftarrow$ true
Obj: fixed-sail-equip-cost(ves,pf,pt)
$-$ $(x^E - x^B)\cdot$var-sail-equip-cost(ves,pf,pt)
Con: min-time(ves,pf,pt) $\leq x^E - x^B \leq$ max-time(ves,pf,pt)
$\wedge$ $h^B_{\texttt{ves}} \leq x^B$ $\wedge$ $\Delta^{SOS}_{\texttt{ves}} = 0$

Figure B.5: The LTOP sail-equipment action for vessel ves $\in V$ from port pf $\in P$ to port pt $\in P$.

sail–on–service(ves, port, sos)
Pre: vessel-at(ves) = port $\wedge$ vessel-state(ves) = T
$\wedge$ $\neg$using-sos-or-se(ves) $\wedge$ sos-open(sos)
Eff: vessel-at(ves) $\leftarrow$ end-port(sos)
$\wedge$ using-sos-or-se(ves) $\leftarrow$ true $\wedge$ sos-open(sos) $\leftarrow$ false
Obj: fixed-sos-cost(port, sos)
Con: $\Delta^{sos}_{\texttt{ves}} =$ sos-dur(port,sos)
$\wedge$ $h^B_{\texttt{ves}} \leq$ sos-start-time(port,sos)
$\wedge$ $x^B =$ start-time(port, sos) $\wedge$ $x^E =$ end-time(sos)

Figure B.6: The LTOP sail-on-service action for vessel ves $\in V$ using start port $\in P$ on sos $\in S$.